中科曙光 Sugon 瑞翼教育

I9000

大数据分析管理平台

使 用 手 册

曙 光 瑞 翼 教 育 合 作 中 心

声明

本手册的用途在于帮助您正确地使用曙光瑞翼教育合作中心产品(以下称“本产品”)，在安装和第一次使用本产品前，请您务必先仔细阅读随机配送的所有资料 ,特别是本手册中所提及的注意事项。这会有助于您更好和安全地使用本产品。请妥善保管本手册，以便日后参阅。

本手册的描述并不代表对本产品规格和软硬件配置的任何说明。有关本产品的实际规格和配置，请查阅相关协议、装箱单、产品规格配置描述文件，或向产品的销售商咨询。

如您不正确地或未按本手册的指示和要求安装、使用或保管本产品，或让非曙光瑞翼教育合作中心授权的技术人员修理、变更本产品，曙光瑞翼教育合作中心将不对由此导致的损害承担任何责任。

本手册中所提供照片、图形、图表和插图，仅用于解释和说明目的，可能与实际产品有些差别，另外，产品实际规格和配置可能会根据需要不时变更，因此与本手册内容有所不同。请以实际产品为准。

本手册中所提及的非曙光瑞翼教育合作中心网站信息，是为了方便起见而提供，此类网站中的信息不是曙光瑞翼教育合作中心产品资料的一部分，也不是曙光瑞翼教育合作中心服务的一部分，曙光瑞翼教育合作中心对这些网站及信息的准确性和可用性不做任何保证。使用此类网站带来的风险将由您自行承担。

本手册不用于表明曙光瑞翼教育合作中心对其产品和服务做了任何保证，无论是明示的还是默示的，包括（但不限于）本手册中推荐使用产品的适用性、安全性、适销性和适合某特定用途的保证。对本产品及相关服务的保证和保修承诺，应按可适用的协议或产品标准保修服务条款和条件执行。在法律法规的最大允许范围内，曙光瑞翼教育合作中心对于您的使用或不能使用本产品而发生的任何损害(包括，但不限于直接或间接的个人损害、商业利润的损失、业务中断、商业信息的遗失或任何其他损失)，不负任何赔偿责任。

对于您在本产品之外使用本产品随机提供的软件，或在本产品上使用非随机软件或经曙光瑞翼教育合作中心认证推荐使用的专用软件之外的其他软件，曙光瑞翼教育合作中心对其可靠性不做任何保证。

曙光瑞翼教育合作中心已经对本手册进行了仔细的校勘和核对，但不能保证本手册完全没有任何错误和疏漏。为更好地提供服务，曙光瑞翼教育合作中心可能会对本手册中描述的产品软件和硬件及本手册的内容随时进行改进或更改，恕不另行通知。

目录

插图目录

一、I9000 大数据分析平台简介

1. 引言

每个企业都可以通过分析所有可用的数据来获得竞争优势、提高运营效率并改善客户参与度。然而，调查显示，大多数企业无法或不愿意部署新的基础设施从而进行高级分析，因为他们缺乏可靠的投资回报率，没有掌握先进的技术，并且搭建基础设施太复杂，还要承担数据安全漏洞的风险。所以，需要一款产品来帮助企业完成这些复杂的任务，从而降低企业的执行难度。

目前市面上的产品有一部分为高级分析用例提供了功能，例如 ETL、数据仓库和商业智能。一部分产品则只有传统的分析功能，如报告、仪表板和统计。还有一些产品是分析人员可以在大型数据集上查询（如 SQL）的交互工具。但没有任何产品为复杂的大数据分析（将以形成定制应用程序的方式交付结果）提供解决方案。

所谓的大数据高级分析可以为有需求的企业提供处理行业特定数据、领域特定域分析和企业特定应用程序的定制解决方案。这种由大数据分析驱动的定制应用程序需要一个更高效的流程来生成，并将其部署在可扩展的基础架构上。

这也就是大数据生态系统（I9000）出现的意义，I9000 已经自下而上进行了彻底设计，旨在减少大数据高级分析的复杂性。I9000 使大数据和大数据分析随时可用于定制解决方案以及高效的开发和部署。它提供了一个统一的平台，数据科学家、应用程序开发人员和系统操作员可专注于高价值的任务，而不用花时间在低价值的任务上，例如，维护辅助工具集并将它们以安全的方式与数据源连接。

I9000 还支持数据科学家与应用程序开发人员之间的大规模协作。

2. 概述

大数据生态系统（I9000）是一个大数据分析平台，它提供了一个丰富的辅助工具包，并针对性能和安全进行了优化，可用于对大数据进行高级分析，加快由大数据分析驱动的云原生协作的创建速度。可以使用 GUI 或基于文本的 shell 对任何类型的原始数据、实时流式处理或批处理数据进行交互式分析、建模和算法处理流程。这些模型和流程可用于批处理或集成到应用程序中。

I9000 支持任何的 web 语言（例如 Python、Java、PHP、Ruby、Javascript）、可在 HTTP 上使用的 REST API 以及可用于服务器本地访问的 Python API。

I9000 可以在大多数数据存储和文件系统上运行，包括能够实现数据共享（具有安全性）的集群联合。I9000 中的集成操作管理工具支持从上到下进行监控和控制。为了支持信任，I9000 安全遵循分层安全和深层防御原则提供透明的加密和解密，以及基于各种身份验证机制和保证级别的细粒度访问授权。

2.1 使用 I9000

I9000 的设计考虑了三种类型的用户：数据科学家、应用程序开发人员和系统操作员。它仔细考虑了每个用户的具体需求以及针对安全性、性能和协作进行优化。

2.1.1 数据科学家

I9000 为数据科学家提供了可扩展的工具，可扩展的算法和强大的引擎来训练和部署大数据驱动的分析模型。特别是，它提供了执行常见任务所需的工具和服务，重点是支持与大数据相关的复杂性，并使高级大数据分析更容易交付给应用程序开发人员，以便在其应用程序中使用。

图片 1 在 I9000 中数据工作的简单流程

在 I9000 中，数据科学家能够使用到熟悉的界面（如 iPython 笔记本、RStudio，H2O Flow 或 Eclipse IDE）来构建和训练大数据模型。I9000 还提供了图形计算、深度学习和“经典”机器学习算法。每个算法都是开源算法并且几乎所有算法都是并行的，可以在分布式处理系统（如 Apache Spark 或 Apache Hadoop）上执行，某些情况下，也可以进行 CPU 和 GPU 硬件加速。

图片 2 在 I9000 中使用数据的关键组件

2.1.2 应用程序开发人员

应用程序开发人员可以基于开源 Cloud Foundry 立即访问多语言应用程序平台。I9000 提供对 Python、R、Java、Scala、Node.js、PHP、Go 等运行时的预配置支持，并且能够与可动态绑定的服务和 API 表达式结合使用，从而可大大减少应用程序开发人员的开发时间，简化与数据科学家所开发的数据分析功能的集成。

I9000 中还实现了很多常用的提取协议(如 MQTT、WS 和 REST)以及基于 Apache Kafka 和 RabbitMQ 的消息队列。这样应用程序开发人员便可以轻松连接到数据处理服务，如 Apache Spark。I9000 还包含一个基于 Akka 的框架，称为 Gearpump，该框架能够对几乎实时的案例（例如，物联网警告方案）进行分布式和低延迟数据处理。

2.1.3 系统操作员

I9000 为系统操作员提供了集成的自助服务硬件和软件堆栈，并针对安全性、可扩展性和性能进行了优化。它减少了通常与云基础架构中的用户配置自助服务环境相关的许多复杂性。

I9000 包括强大的部署和自动化工具，能够快速供应和编排许多异构环境。使用简单的管理界面和服务的开放框架，系统操作员可以扩展 I9000 中的模块化架构并适应其特定的环境。I9000 还包括专注于平台操作的功能，例如系统遥测、安全管理和配置。

2.2 平台架构

I9000 是一个多租户平台，旨在简化和加速端到端分析应用程序的交付。其松散耦合且分层的架构可在定制解决方案交付方面实现更大的灵活性。它并非从头开始，部署大量不同的工具、软件包和服务，而是提供一个可扩展的环境，将很多开源组件组合到一个集成的平台中。该平台提供 API、服务和可扩展性，以支持数据科学家和应用程序开发人员对位于任何位置任何大小的任何数据进行各种分析的需求。此外，I9000 还提供管理工具和服务以便从上到下控制和监控操作。

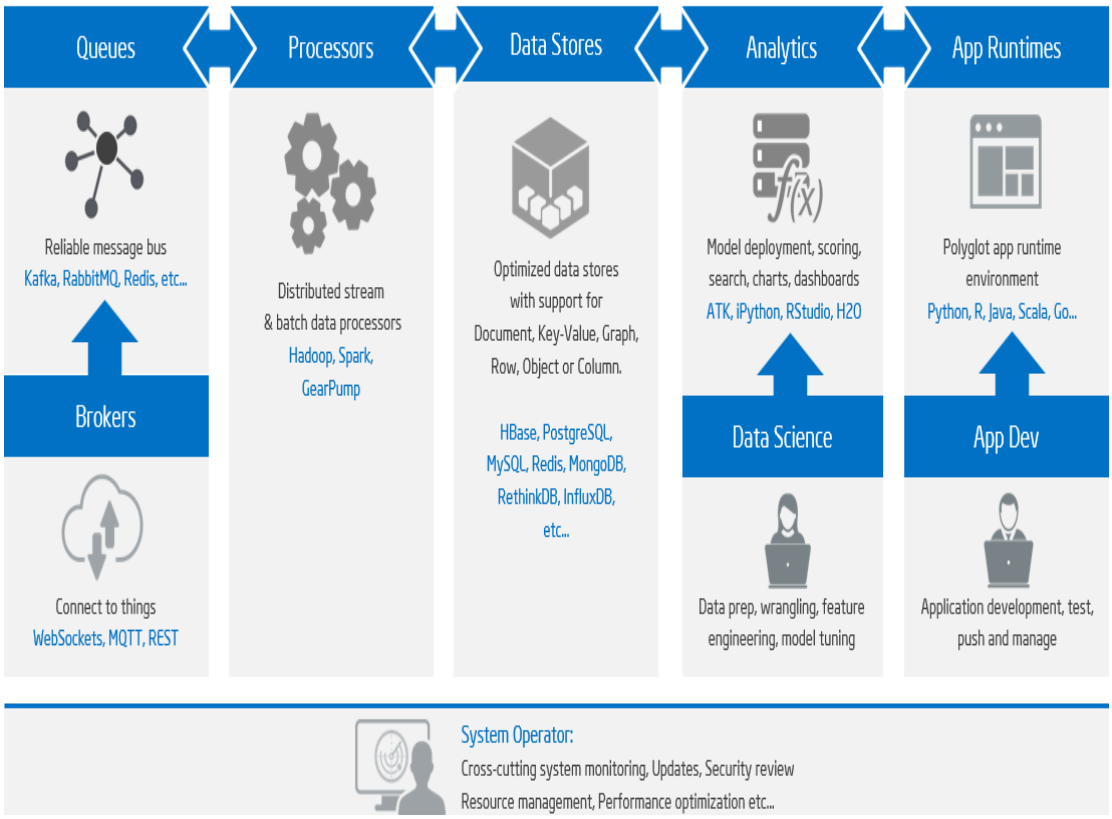
图片 3 平台架构

数据层组合了所有大数据管理系统的基本组件：分布式文件系统和分布式处理框架。此外，数据层还有一系列数据存储，每个存储都针对某个特殊的结构进行了优化：键值、关系数据库关系系统、面向文档、柱状、图形、时间序列及其他。虽然 I9000 中支持的数据存储数量不断增长，但该层的确切组成是可配置的。

分析层包括分析工具和 API 服务器，可将分析工具中的功能调用转换成用于数据整理和机器学习的支持算法。其插件架构允许系统操作员扩展 I9000 功能，并自动公开用户可访问的 API 用于新增的功能。在这一层中，I9000 还利用其合作伙伴的很多技术，包括 H2O 和 DeepLearning4Java。

基于 Cloud Foundry，**应用层**显示各种运行时、用于动态服务绑定的消息传递框架或连接器、用于轻松访问的服务代理以及用于启用服务发现的服务目录。此外，应用层还支持

一个名为“buildpacks”的扩展结构。当开发人员发布他们的应用程序时，Cloud Foundry 会自动检测需要哪个 buildpack 并将其安装在需要运行应用程序的 Droplet Execution Agent (DEA) 上。



图片 4 逻辑组件

在平台架构的每一层，都已进行性能优化，目的是最大程度提高分析操作的速度。与此同时，通过芯片增强安全性，从而确保数据和操作受到保护。

2.3 其他功能和特点

2.3.1 数据平台设置和连接

I9000 使用的数据层基于 Cloudera Distribution of Hadoop (CDH) 并结合了常用大数据管理平台的核心基本组件：Hadoop Distributed File System (HDFS) 和

Distributed Processing Framework（Map Reduce 和 Apache SPARK）。

此外，数据层可以用一系列数据存储来扩充，这些数据存储可以由 I9000 以微服务的形式提供。每个微服务可以针对特定结构（例如，键值、关系数据库管理系统、面向文档、柱状、图形、时间序列等）进行优化。I9000 中支持的数据存储微服务的数量继续扩大，以支持新的用例。

虽然单片应用程序通常使用单一的逻辑数据库来保存数据，但是微服务架构风格允许每个服务管理自己的数据库，作为相同数据库技术的不同实例或作为不同的数据库系统。根据这个通晓多种语言持久性的原则，可以在 I9000 上部署各种数据存储。虽然首选基于分布式文件系统（HDFS）的可扩展数据存储，但 I9000 也包括单节点但却非常流行的关系数据库管理系统（RDBMS），如 MySQL 和 PostgreSQL。

2.3.2 数据流（管道）

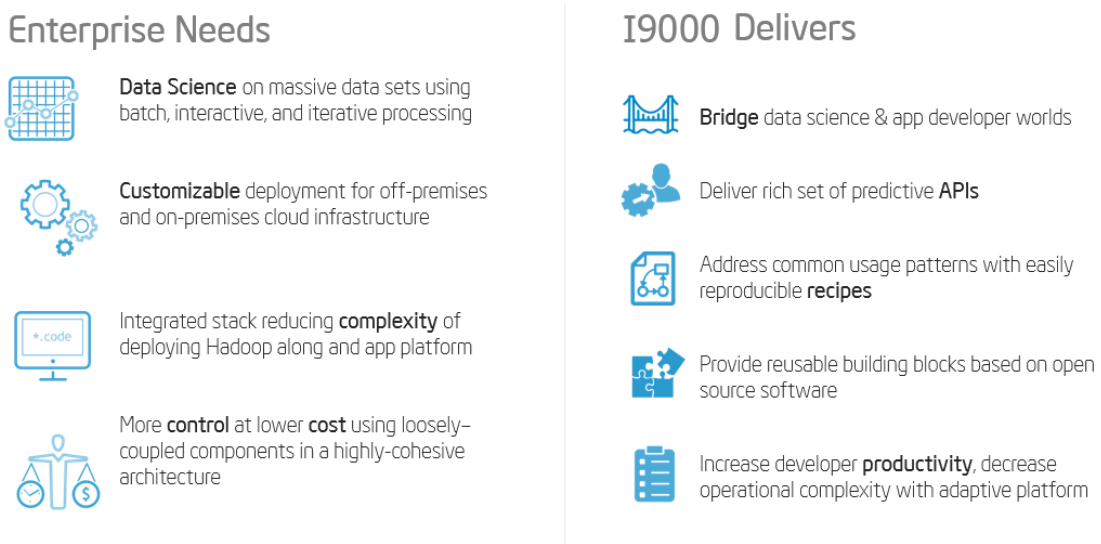
I9000 能够创建来自简单连接器灵活连接的多个类别中的复杂服务管道。例如，服务可以通过 MQTT 来提取流，使用 Gearpump 服务转换原始字段，作为主题持久保存到 Kafka 队列以及使用以前存储在 Hadoop 群集中的历史数据进行复杂模型的训练以及对结果进行评分。

I9000 还包括一个评分管道，数据科学家在模型开发期间使用该管道可轻松部署数据管道，然后应用程序开发人员使用这些数据管道构建分析驱动的产品和服务。该评分管道利用 Python User Defined Functions（UDF）来支持在与评分引擎相同的流程中部署数据处理和特征提取。I9000 提供的这种组合能力可在应用程序开发人员部署到产品时更高效、更灵活地交付评分引擎的预测结果。

2.3.3 I9000 市场

I9000 包含一个丰富的市场，在该市场中可以根据需求轻松集成和设置工具和服务。这个 I9000 市场可通过一个简单的、基于浏览器的界面从服务目录中选择组件以及创建每个组件的实例。应用程序开发人员、数据科学家以及系统操作员能够灵活选择他们要用于提取、存储以及处理数据的工具和服务。此外，系统操作员还可以在他们的 I9000 实例中向 I9000 市场添加服务，无需识别和策划用于交付新的分析应用程序所需的关键工具和库，从而节省了用户的时间。所有这一切都是在一个可安全交付这些组件且考虑性能的协作环境中完成。

2.3.4 I9000 交付方式



图片 5 平台交付方式

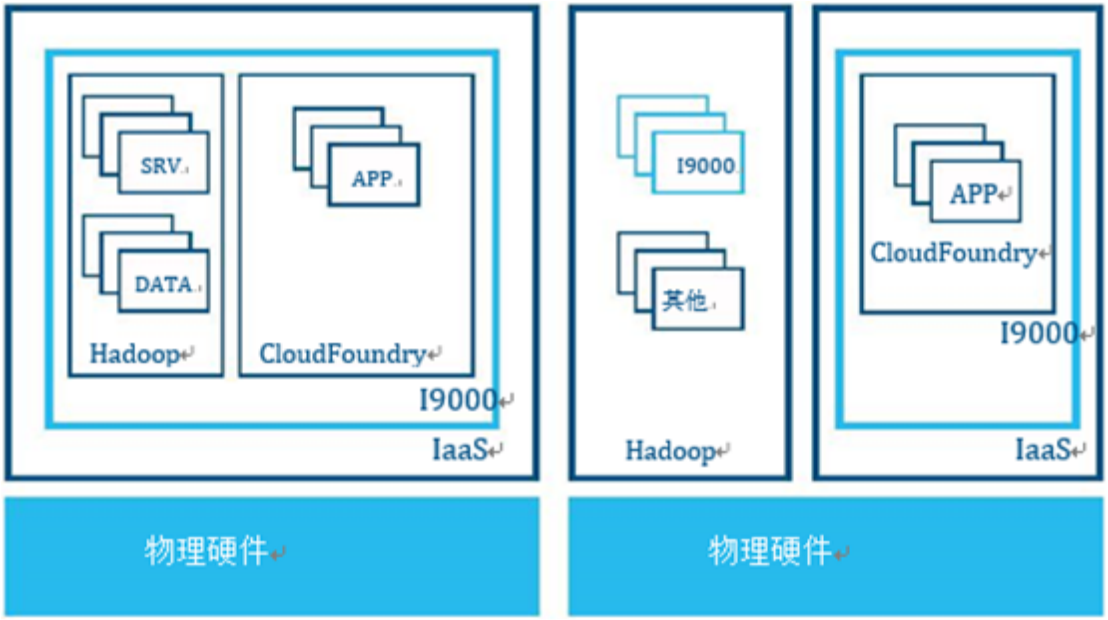
2.4 可扩展性

I9000 提供一个可扩展的框架，该框架允许将新的命令和算法以插件的形式添加到应用程序中。它们采用 Scala 编写，Scala 是一种类似于脚本的函数编程语言。插件不需要

创建者对 REST 服务器、执行引擎或它的任何库有深入的了解。该框架会创建通过 REST API 调用插件所需的代码（采用 Python）并且会生成 Python 客户端代码，从而让创建者专注于执行逻辑。

2.4.1 部署模型

I9000 支持两个部署模型。已经在其数据中心（虚拟部署或部署在裸机上）部署 Apache Hadoop 的组织可以再次使用现有群集或选择 I9000 来部署。



图片 6 部署模型

3. 了解 I9000 控制台

本教程将带您了解大数据分析平台（I9000）的主要控制台，以便您熟悉其中的内容。了解控制台之后，您便可以深入了解您感兴趣的特定领域。

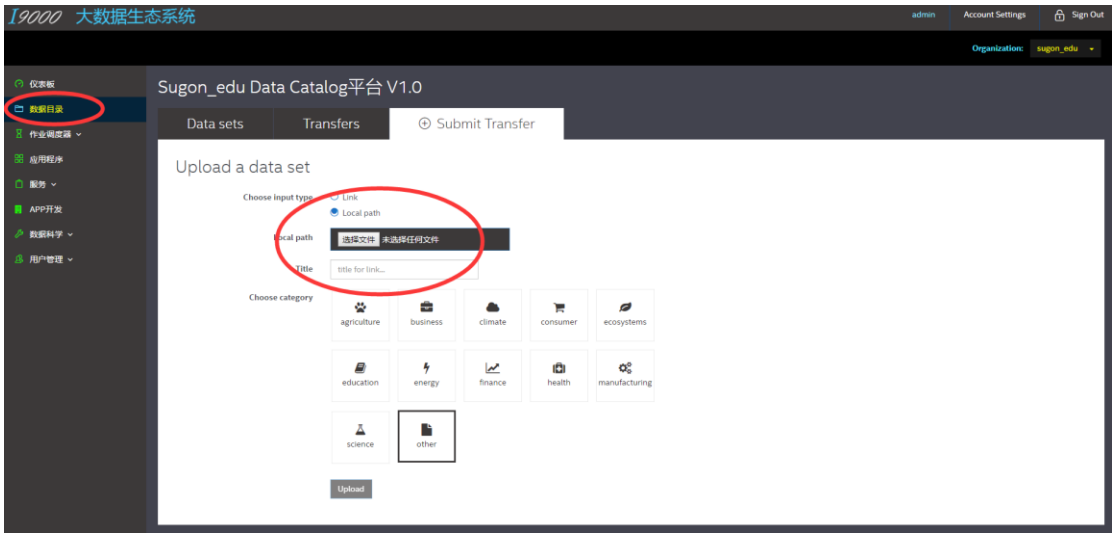
3.1 数据获取

I9000 具有三种收集数据的方式：手动上传数据、从数据源批量或定时获取数据以及流

数据获取。在这里只是简单介绍，详细说明见本操作手册后面部分：四、I9000 中的数据获取。

3.1.1 上传数据

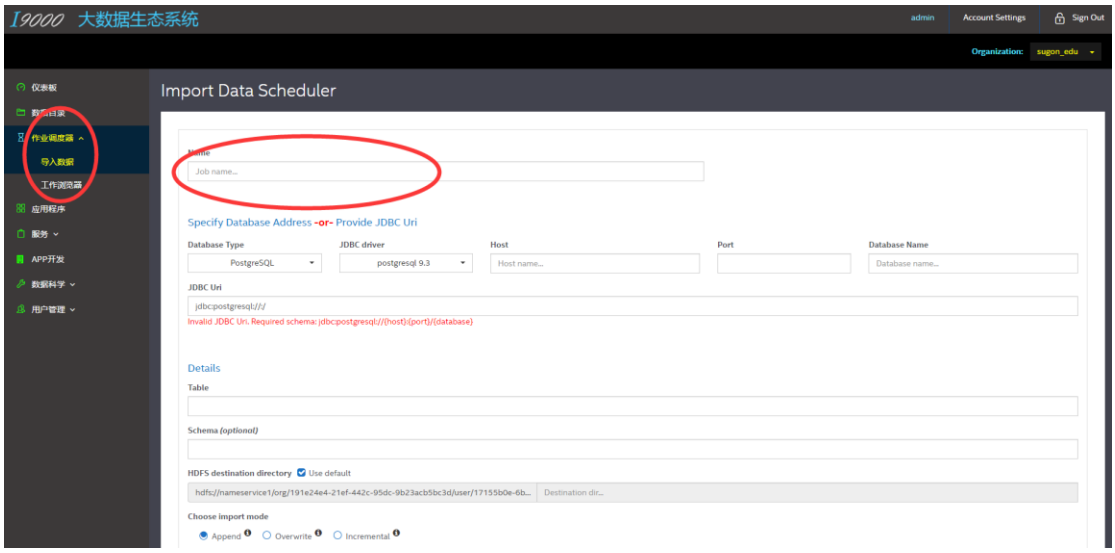
I9000 的“数据目录”选项卡支持上传导入各种格式的数据（参见下图）



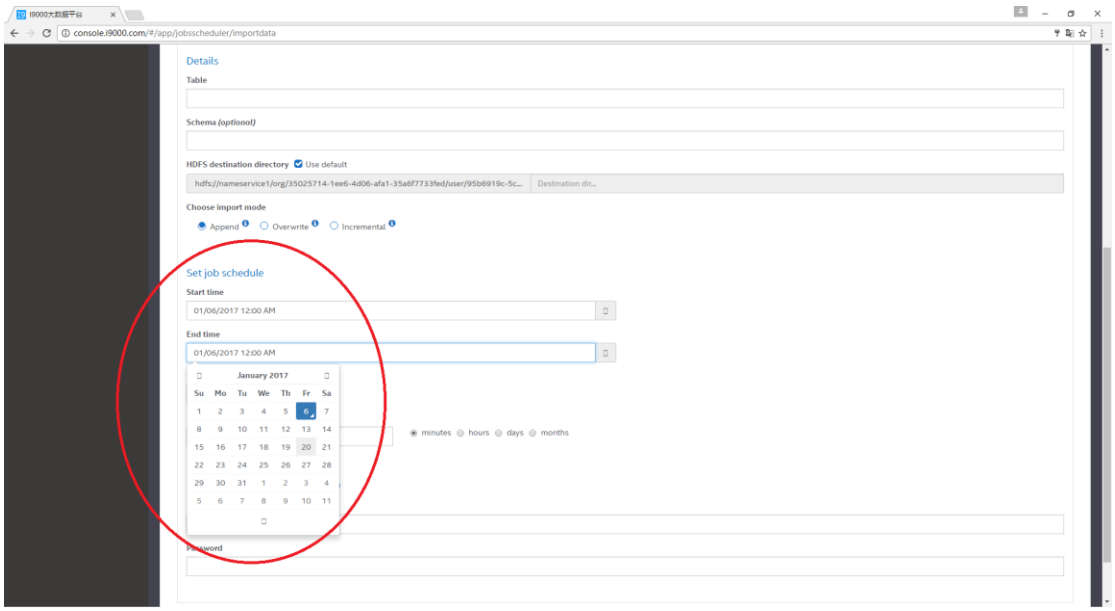
图片 7 数据目录

3.1.2 用作业调度器获取数据

I9000 的“作业调度器”采用一种特定的数据格式定时从数据库中导入数据（参见下面两个截图）



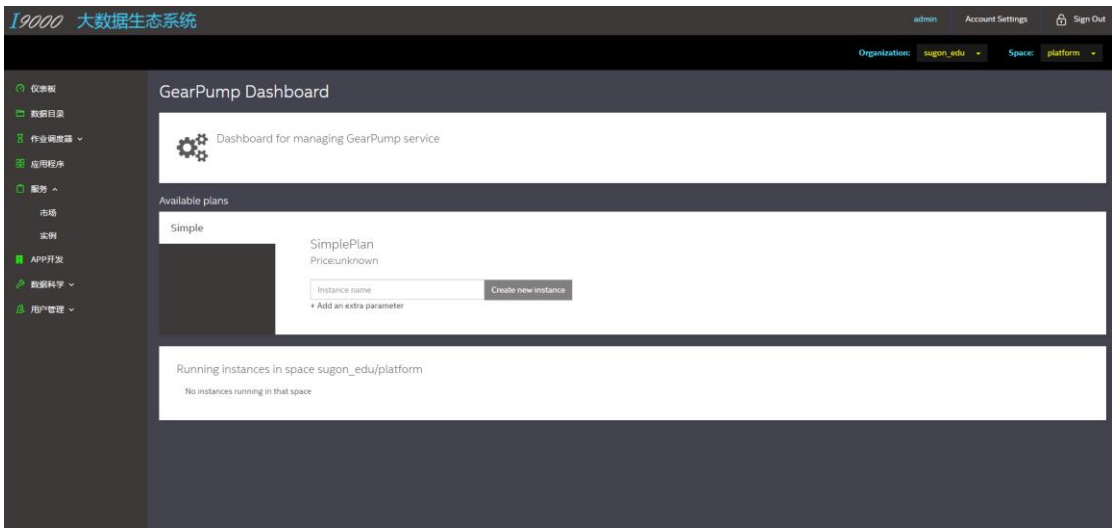
图片 8 作业调度器



图片 9 日期控件

3.1.3 流数据获取

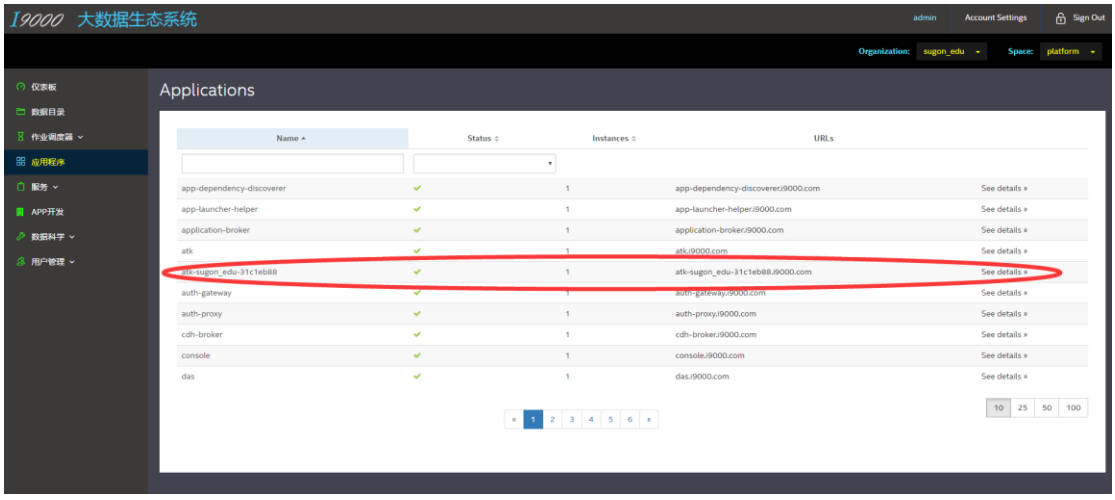
您可以配置通道，以便使用服务（例如 Apache Gearpump）从实时数据源导入流式数据（如下所示）



图片 10 GearPump 工具

3.2 应用程序

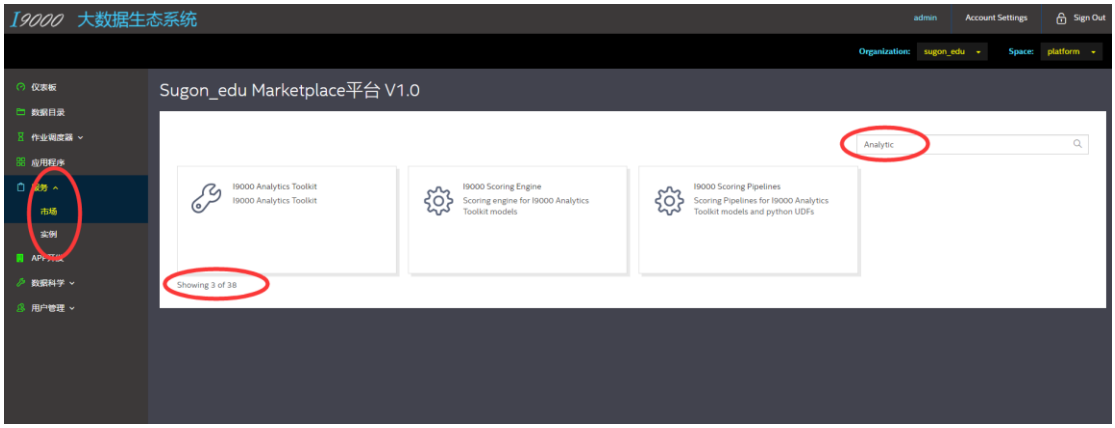
选择并运行存储的应用程序以便通过“应用程序”选项卡显示您的分析。



图片 11 应用程序

3.3 服务

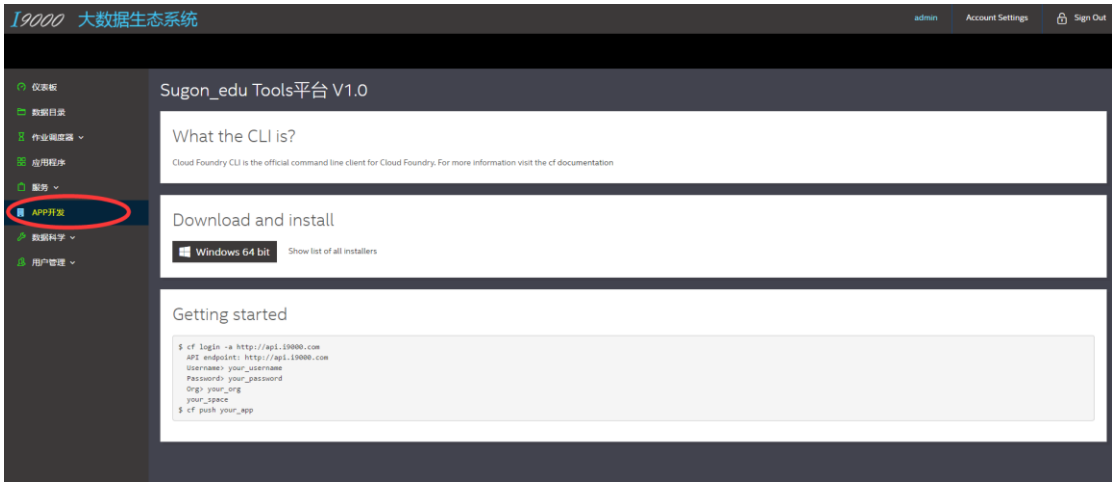
I9000 包括一个可帮助您进行数据分析、应用程序开发和系统监控的服务市场。下面的屏幕截图显示了从 38 个受支持服务的完整套件中选择搜索术语“Analytic”的服务。



图片 12 服务

3.4 APP 开发

“APP 开发”选项卡可使用 CF CLI 命令将您自己的应用程序推送到 I9000 平台。示例 CLI 命令可帮助您开始这些操作。如果您运用 CLI 方法，则会让您安装应用程序。安装到该平台之后，您便可以通过“应用程序”选项卡运行您的应用程序。

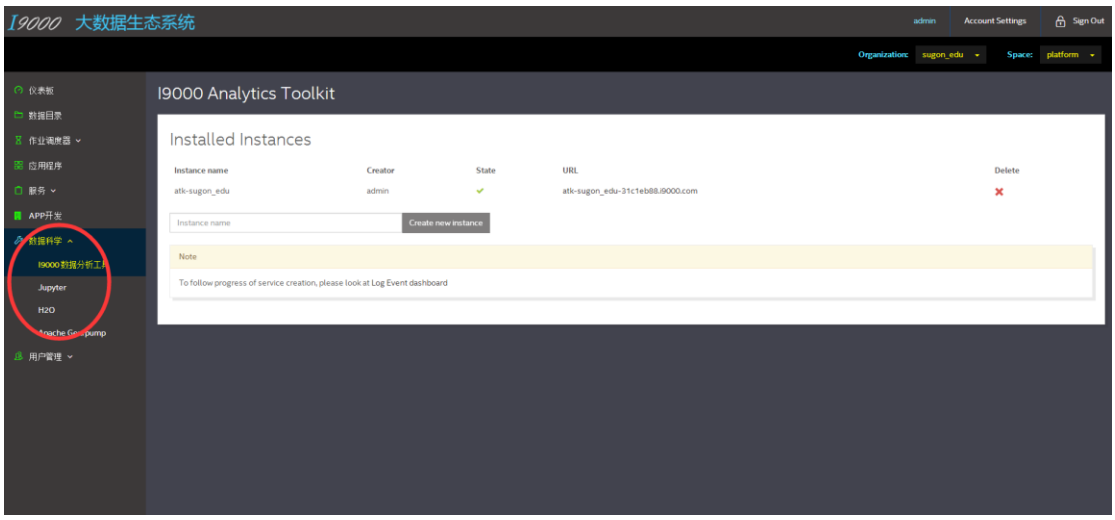


图片 13 APP 开发

3.5 数据科学

数据科学功能提供了对深入的机器学习算法(这是定制的分析工具包的一部分)的访问。

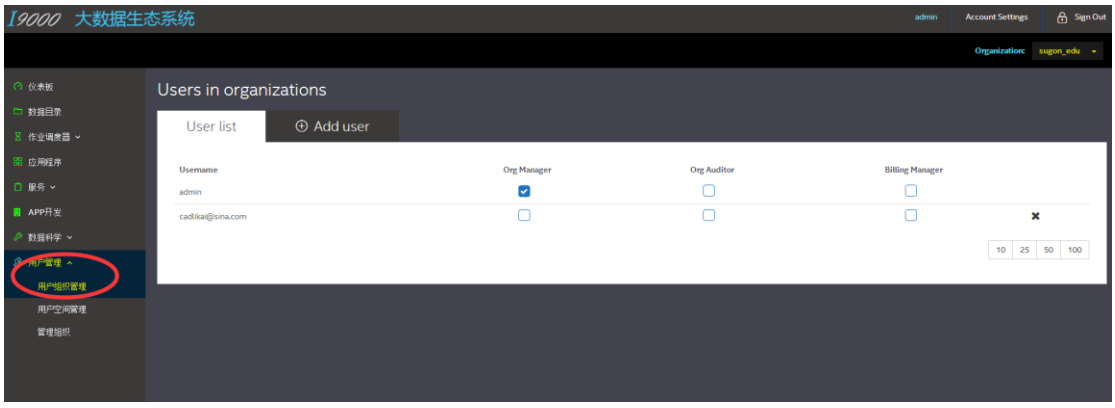
目前主要有 I9000 数据分析工具、Jupyter、H2O 和 Apache Gearpump。



图片 14 数据科学

3.6 用户管理

“仪表板”和“用户管理”功能可管理 I9000，以便帮助您的团队实现其目标。您可以通过“仪表板”选项卡管理系统、通过“用户管理”选项卡管理用户（ “管理组织用户”屏幕如下所示）。



图片 15 用户管理

现在您已熟悉 I9000 平台结构 ,接下来您便可以更加深入地探索大数据分析平台 I9000

的强大功能了。

4. 平台安全功能

4.1 简介

大数据分析平台（I9000）是一个从设计之初就考虑了安全的大数据分析平台，主要有两个方面：

(1)它提供了强大的安全机制并简化了它们的使用；

(2)“大数据分析”的推动者，在分析数据的同时保持数据的机密性和隐私性。

I9000 架构围绕两个核心组件：Cloudera Hadoop (CDH) 和 Cloud Foundry (CF)。I9000 以无缝的方式集成了这两个组件。其余 I9000 元素构建在这些核心组件之上或为它们提供服务，如下所示：

Amazon Web Services (AWS) 或 OpenStack：操作 CDH 和 CF 所需的基础设施即服务（IaaS）层。它为虚拟机创建，存储和网络机器提供 API。

Cloudera Distribution of Hadoop (CDH)：一个大数据软件堆栈。

Cloud Foundry (CF)：用于开发和托管云应用程序的平台。

I9000 集成和定制服务：该层由用户用于创建应用程序的不同类型的服务组成。这些服务包括各种组件类型，包括原始的开源服务，如 MySQL 或 Redis，以及内部开发的组件，如“数据目录”。

最终用户应用程序：该层使得应用程序开发人员能够利用平台服务在 I9000 堆栈之上构建自定义应用程序。

4.2 安全机制

平台安全包括几个明确定义的机制，用来保护应用程序以及数据：

身份验证：I9000 继承和扩展了 CloudFoundry（CF）中支持的身份验证机制（将这些机制与 Hadoop 中的身份验证机制集成在一起）。CF 使用 OAuth 2.0，Hadoop 使用 Kerberos。I9000 通过一个自定义的安全网关将这两者集成在一起，从而允许 CF 应用程序通过 OAuth 2.0 对用户进行身份验证并将 OAuth 令牌转换成 Kerberos 接受的服务票证。这为用户提供了跨数据和应用程序层的单点登录（SSO）体验。

授权：I9000 授权给 CF 以及各个数据组件。更多详细信息如下所述。

隔离：I9000 使用 CF 中的 Warden 容器来强制进行应用程序隔离。I9000 能够通过根据其父组织及其访问范围限制用户的可见性来实现数据集的隔离。I9000 使用 OAuth 令牌与数据集的元数据（指示访问范围）组合来进行此决定。

I9000 还解决了以下数据和分析安全问题：

机密性：I9000 通过启用具有高性能奇偶校验的加密来保护数据的机密性，这样便可以随时对静止和运动的所有数据进行加密，而不会损失应用程序性能。I9000 用户可以配置 Hadoop 透明数据加密。使用透明加密，从 Hadoop 文件系统（HDFS）目录读取和写入的数据会被透明地加密和解密，而不需要更改用户应用程序代码。更多详细信息可以在 Hadoop 文档页面上找到。

隐私性：保护隐私的事务已经计划处理。

4.3 用户管理

I9000 使用 CF 的用户帐户和身份验证服务器（UAA）对用户进行身份验证并继承 CF 的访问控制系统，在该系统中用户被分配到特定的组织和空间并且可以在这些组织单位中拥有各种角色。

组织：组织是个人或多个协作者可以拥有和使用的开发帐户。所有协作者都可以通过用户帐户访问组织。组织中的协作者共享资源配额计划、应用程序、服务可用性以及自定义域。

空间：每个应用程序和服务都限定在一个空间之中。每个组织至少包含一个空间。空间为用户提供对共享位置的访问，以便进行应用程序开发、部署和维护。每个空间角色仅适用于某个特定的空间。

4.4 角色和权限

用户可以拥有一个或多个角色。这些角色的组合定义了该用户在该组织以及在该组织的特定空间中的总体权限。组织级别的角色包括 Org Manager（组织管理员）、Org Auditor（组织审计员）和 Billing Manager（帐单管理员）。空间级别的角色包括 Space Manager（空间管理员）、Space Auditor（空间审计员）和 Space Developer（空间开发人员）。可在此处找到这些角色及其相关权限的描述。

4.5 用户注册过程

若要注册新用户，系统管理员会发送一封电子邮件，邀请用户创建一个用户帐户和一个新的组织。注册过程开始会生成一个安全代码并将其存储在安全代码数据库（Redis 数据库）中。然后“用户管理”服务会使用一个外部 SMTP 服务发送一封邀请电子邮件，其中包含一个含有安全代码的 URL。受邀用户可使用提供的 URL 可访问注册表单。

4.6 审计和日志

I9000 平台提供多个级别的审计功能，如下所示：

4.6.1 云级别

底层的 I9000 IaaS 提供商，如 AWS 和 OpenStack，支持审计和日志记录功能。它们记录基础架构中的每个更改。更多详细信息可以它们各自的站点上找到。

4.6.2 基础架构级别

BOSH 是处理平台中基础架构变化的守护程序，它生成 Cloud Foundry 组件所做的所有更改的历史记录。它显示对平台所做的详细更改集，以及该进程的调试日志。

4.6.3 应用程序级别

Cloud Foundry 允许运营商从应用程序访问事件和日志。事件由平台提供，包括诸如应用程序重新启动等事件，而日志由应用程序自身提供，报告其内部工作。更多详细信息可以在 CF 文档中找到。

Cloudera 提供两种类型的日志。一种用于管理服务，另一种用于在其上运行的 Hadoop 服务。

此外，I9000 还利用 LogSearch 堆栈通过 Kibana 来显示基础架构和应用程序级别日志：一个易于使用的 Web 界面，支持浏览、过滤和记录日志。目前，CF 日志在 LogSearch 和 Kibana 集成中显示。该集成支持 SSO，从而用户能够使用他们的 I9000 帐户凭据登录到该 Web 界面并且能够验证他们的角色访问权限。

二、平台管理

1. 邀请用户加入 I9000 平台

若要邀请新用户加入该平台，请执行以下步骤：

(1) 以系统管理员的身份登录 I9000。

(2) 导航到“用户管理”并选择“适职”选项卡。

(3) 输入您要邀请的用户的电子邮件地址并单击“发送邀请”。

受邀用户会收到一封含有到注册页面链接的电子邮件，在注册页面中用户可以指定他们自己的密码并可以创建/命名他们自己的组织。

1.1 响应邀请

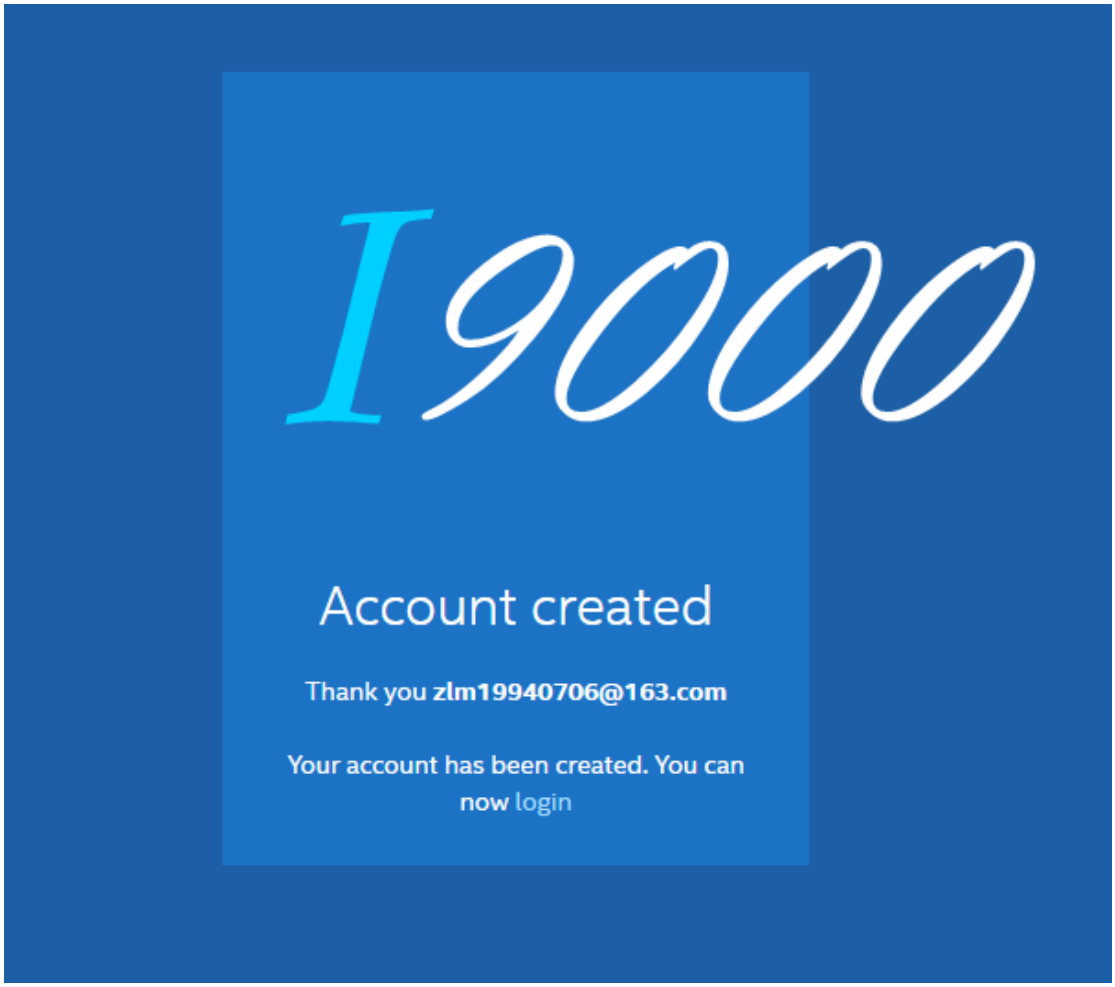
如果您还不是该平台的成员，组织/空间所有者便可以邀请您加入现有组织或空间，也可以邀请您加入该平台。收到电子邮件邀请并单击电子邮件内的链接之后，您便会来到下面的注册页面。

图片 16 注册页面

输入您的密码选择（两次），选中服务条款复选框并单击“创建帐户”。

I9000 会创建您的平台帐户并且您可以访问邀请您加入的组织。

然后 I9000 会显示一个确认屏幕（如下所示）。单击确认屏幕上的登录链接即可登录该平台。（记得保存登录链接以便将来使用）



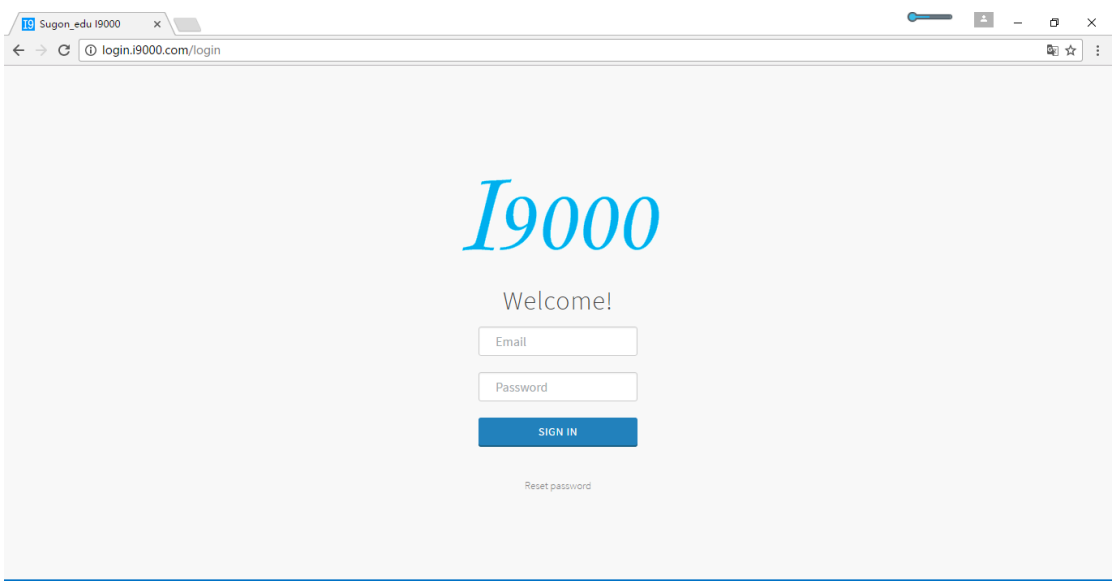
图片 17 注册成功

如果您已拥有 I9000 用户帐户，那么当系统管理员或组织/空间所有者将您添加到组织/空间时，您不会收到电子邮件邀请。对于这些添加，由系统管理员或组织/空间所有者负责通知您。

2. 访问用户帐户

初始访问该平台以及创建组织的请求只能通过系统管理员的邀请才可实现。有关这种类型的平台邀请，请联系您的系统管理员。（对于其他类型的邀请，请参见下一主题，但该主题不包括创建组织的请求。）

拥有 I9000 帐户之后，您便可以使用以下屏幕来登录：



图片 18 登录页面

3. 管理组织/空间和用户

本页向您展示如何在组织下创建新的组织和新的空间。还会展示如何邀请用户加入现有的组织或空间。此外还包含从组织/空间中删除组织和空间以及用户的步骤。

3.1 添加新的组织

您必须是系统管理员才能进行该操作。下面是创建新的组织的步骤：

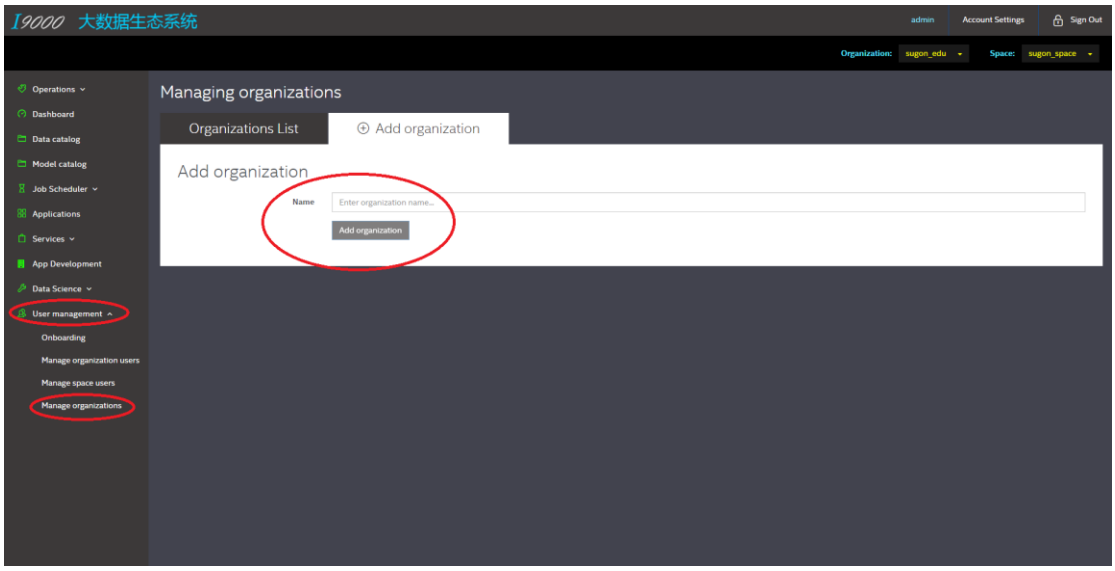
(1) 导航到“用户管理”并单击“管理组织”。

(2) 选择“添加组织”选项卡。

(3) 在“名称”字段中为组织指定名称。

(4) 单击“添加组织”按钮。

(5) I9000 会创建该组织并且会在屏幕右上角显示确认消息。



图片 19 添加组织

3.2 删除组织

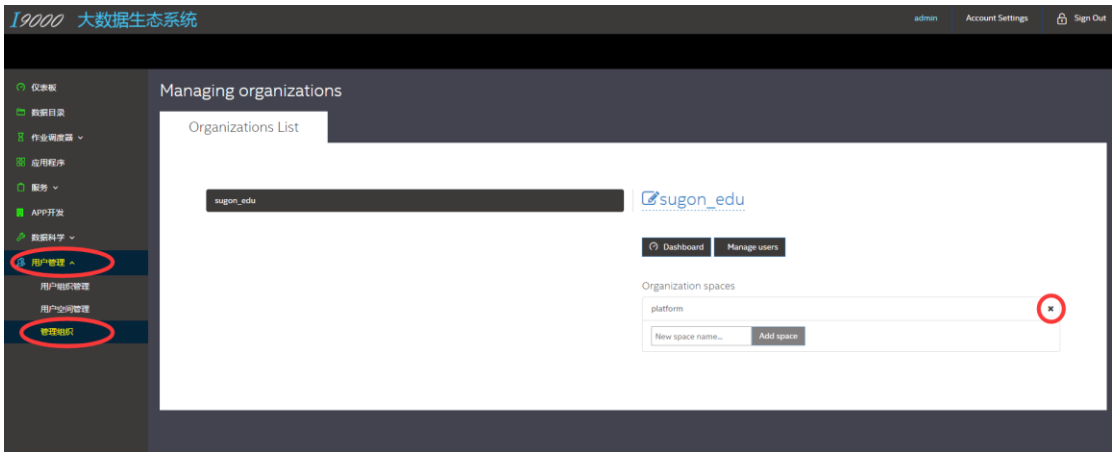
您必须是系统管理员才能进行该操作。下面是删除现有组织的步骤：

(1) 导航到“用户管理”并单击“管理组织”。

(2) 选择“组织列表”选项卡。

(3) 在左侧选择您要删除的组织。

(4) 单击右下角的“删除组织”按钮。



图片 20 删除组织

(5) 确认删除。

(6) I9000 会删除该组织并且会在屏幕右上角显示确认消息。几秒钟之后刷新您的浏览器便会看到屏幕上已反映该删除。

3.3 添加新空间

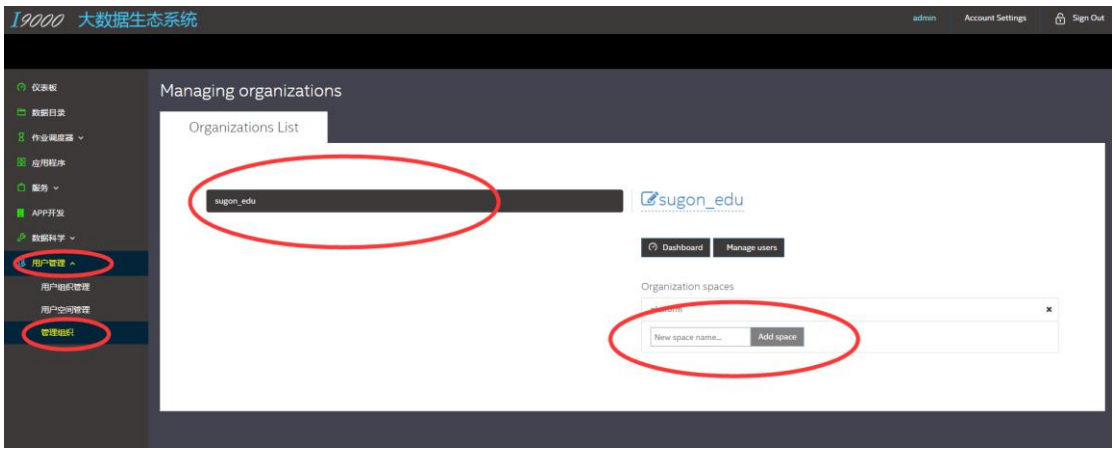
系统管理员可以向任何组织添加空间。其他人可以向他们管理的组织添加空间。下面是在现有组织中添加新空间的步骤：

(1) 导航到“用户管理”并单击“管理组织”。

(2) 在“组织列表”选项卡中选择要向其中添加空间的组织。

(3) 在“组织空间”下指定新空间的名称。

(4) 单击“添加空间”按钮。



图片 21 添加空间

(5) I9000 会创建该空间并且会在屏幕右上角显示确认消息。

您可以通过单击空间名称后面的 x(删除符号) , 然后确认删除来删除您的空间。I9000

会删除该空间并且会在屏幕右上角显示确认消息。几秒钟之后刷新您的浏览器便可以看到屏幕上已反映该删除。

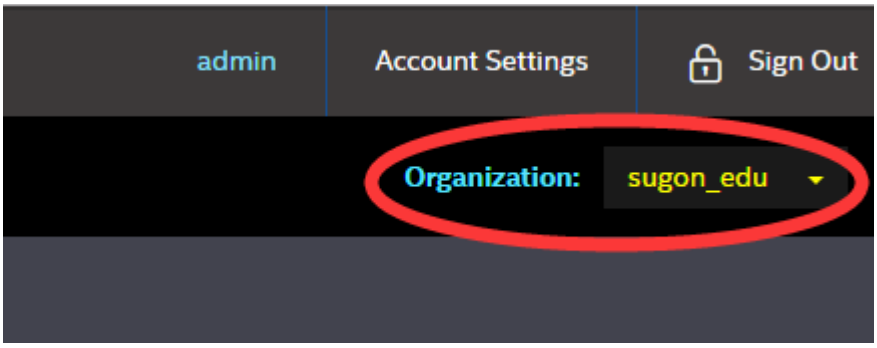
3.4 向组织/空间添加用户

系统管理员可以向任何组织添加用户。其他人可以向他们管理的组织添加用户。如果用户在该平台上还没有用户帐户，那么 I9000 会生成一封该用户的邀请电子邮件。

下面是邀请用户加入现有组织的步骤：

(1) 导航到“用户管理”并单击“管理组织用户”。

(2) 从屏幕右上角的下拉菜单中选择组织。



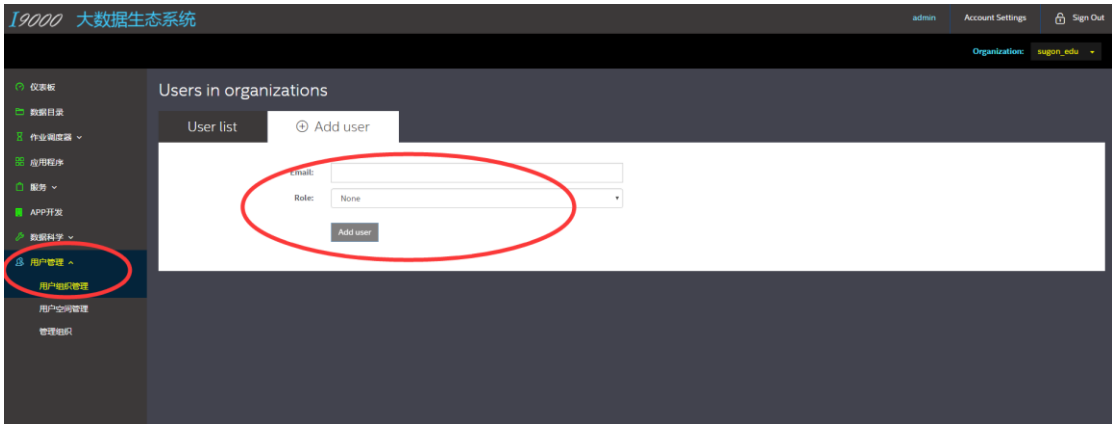
图片 22 选择组织

(3) 选择“添加用户”选项卡。

(4) 输入您要添加的用户的电子邮件地址。

(5) 对于大多数用户来说，无需选择“角色”（默认值 None 适合大多数用户）。对于某些用户来说，需要选择适当的 Manager/Auditor（管理员/审计员）角色。

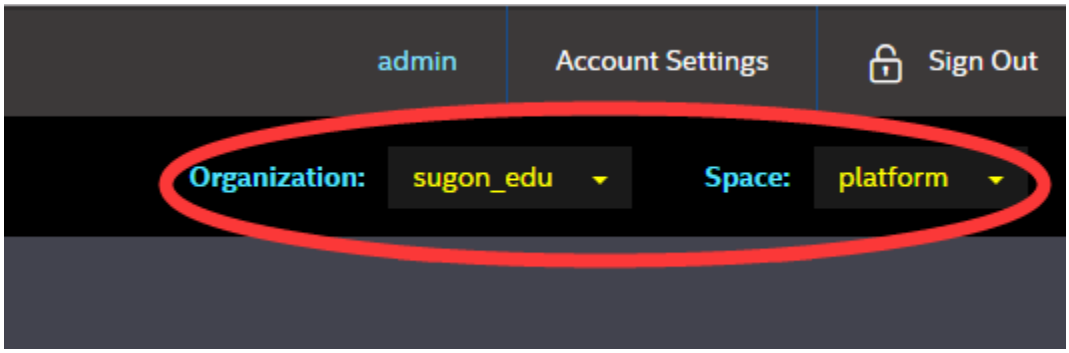
(6) 单击“添加用户”按钮。



图片 23 添加用户

对于现有平台用户来说，I9000 会通知您已向组织添加用户。但您必须将此添加通知该用户，I9000 不发送任何通知。

类似的流程适用于邀请用户加入空间，但您需要在第 1 步中使用“管理空间用户”，然后在第 2 步中从屏幕右上角的下拉菜单中选择组织和空间，如下所示。



图片 24 选择组织和空间

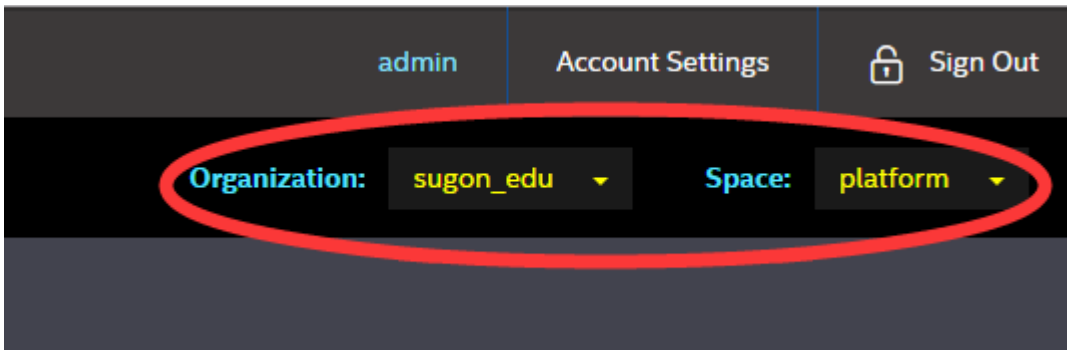
3.5 从组织/空间删除用户

系统管理员可以从任何组织/空间删除用户。其他人可以从他们管理的组织/空间删除用户。

下面是从现有组织（或空间）删除用户的步骤：

(1) 导航到“管理组织用户”（或“管理空间用户”）。

(2) 从屏幕右上角的下拉菜单中选择要管理的组织（和空间），如下所示：



图片 25 选择组织和空间

(3) 单击您要删除的用户旁边的 x（删除符号）。

(4) 确认删除。

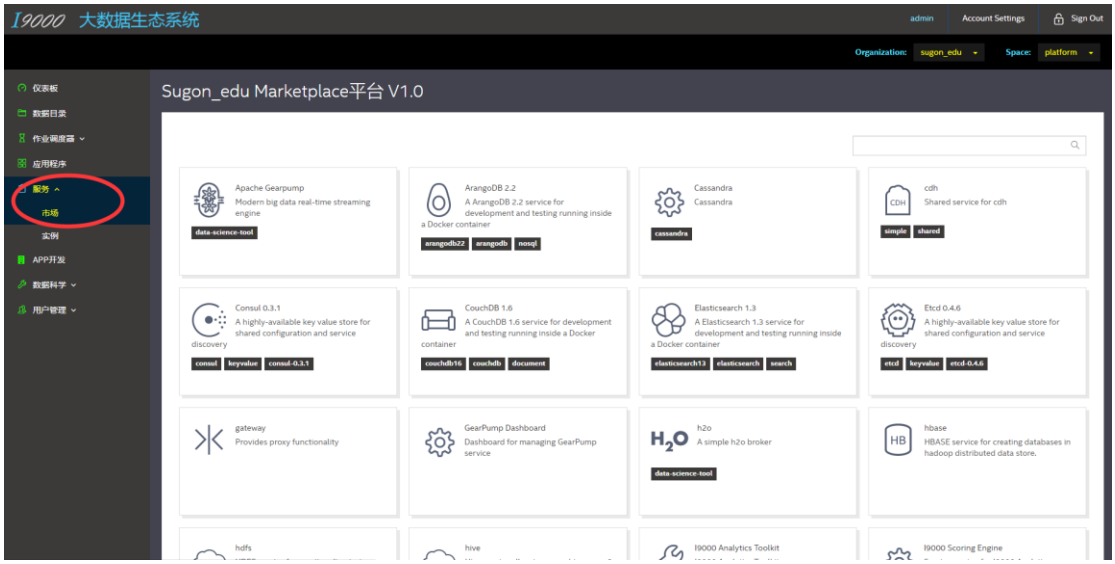
(5) I9000 会删除该组织（或空间）并且会在屏幕右上角显示确认消息。几秒钟之后刷新您的浏览器便会看到屏幕上已反映该删除。

该操作仅仅是从组织或空间中删除用户；他们的用户帐户仍然存在于平台级别。

三、服务市场

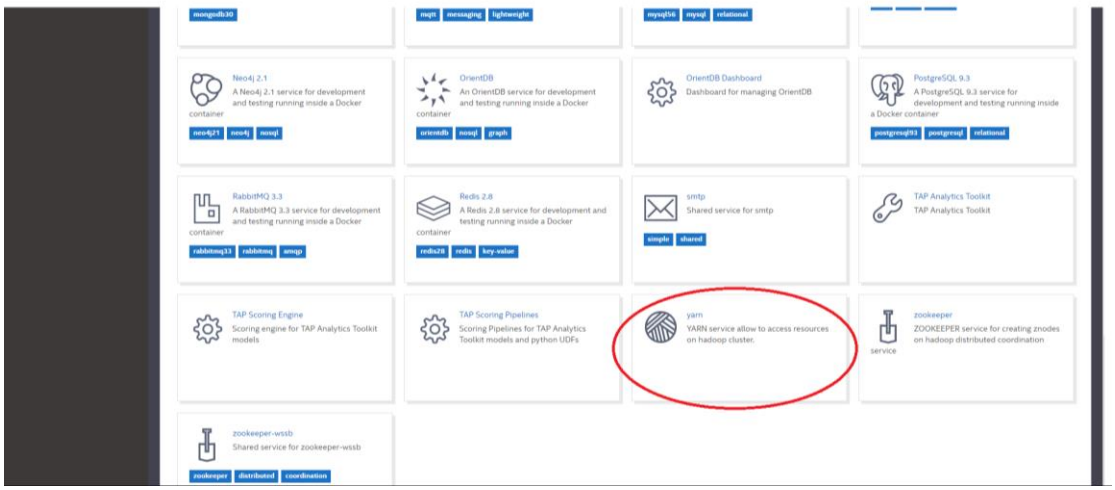
1. 创建服务实例

(1) 从 I9000 控制台，导航到“服务”和“市场”，如下所示。



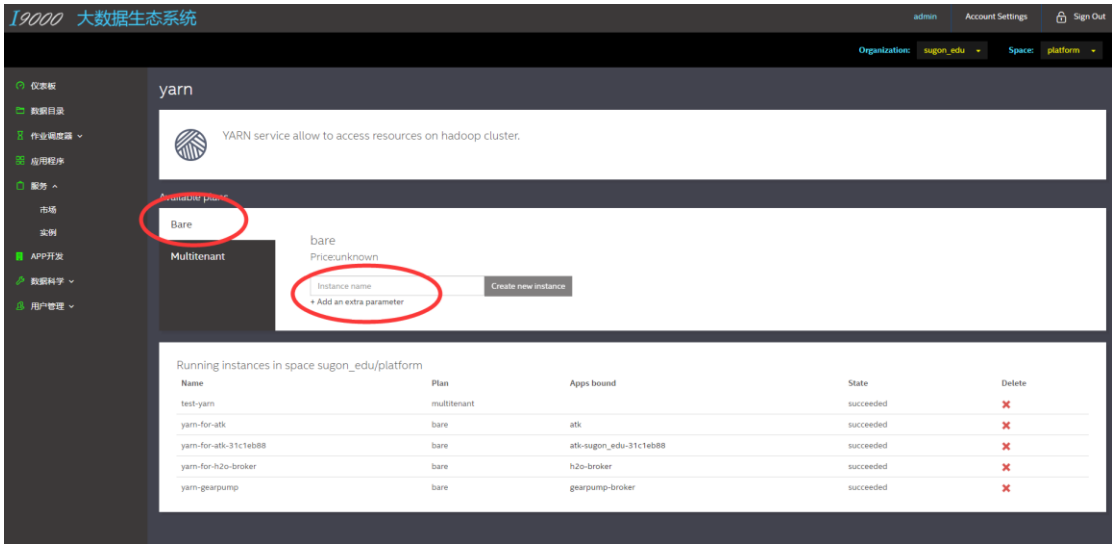
图片 26 市场

(2) 搜索并选择您所需的服务。（对于本例，已选择 Yarn，但这些步骤适用于“市场”中可用的任何服务）。



图片 27 选择服务

(3) 从服务详情页面上，选择所需的计划，然后在“创建新实例”按钮左侧的字段中输入新服务实例的名称。



图片 28 创建服务实例

如果需要键值对作为该服务的参数，请单击“+添加额外参数”以显示一个键值对的字段。输入键和值。（您可以再次单击该链接以便添加另一个键值对。也可以通过单击键值对之后的 x 符号删除键值对。）

(4) 单击“创建新实例”按钮，开始创建您所选择和命名的服务的实例。（只有您为实例输入名称之后，该按钮才可用。）

当安装过程完成之后，您的实例将包含在服务详情页面底部的“正在运行的实例”表中。

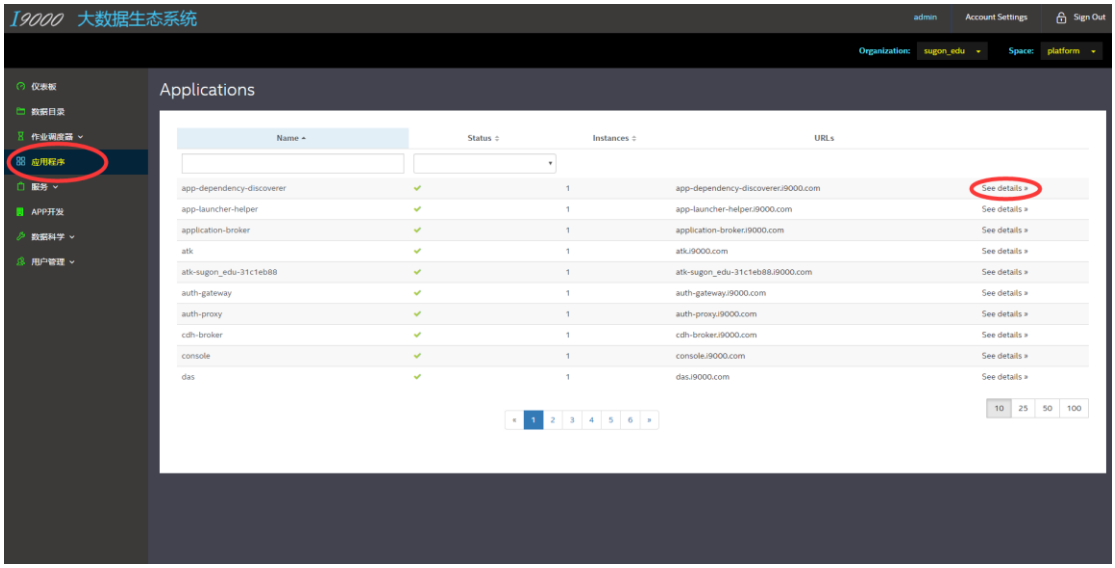
2. 将应用程序绑定到服务实例

为了将应用程序绑定到该服务，必须创建服务的实例。有关说明，请参见“创建新的服

务实例”。

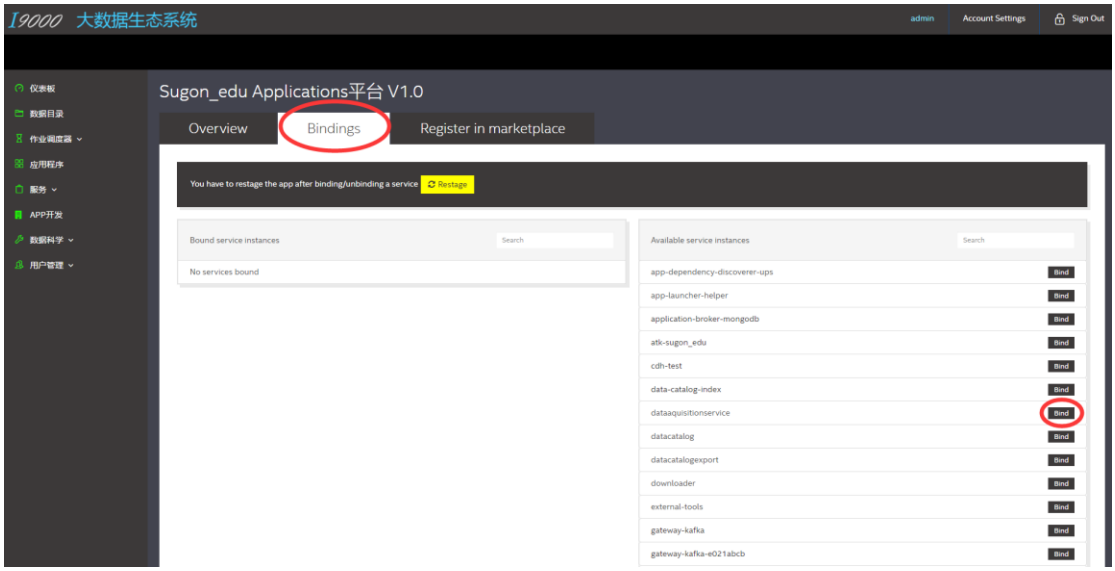
(1) 在 I9000 控制台上，导航到主菜单中的“应用程序”。

(2) 搜索您要绑定的应用程序并单击应用程序行最右侧的“查看详情>>”链接。



图片 29 查看应用程序详情

(3) 在应用程序详情页面上，选择“绑定”选项卡可同时查看已绑定到应用程序的服务以及可为新绑定选择的可用服务的列表。



图片 30 绑定页面

(4) 单击您要绑定到应用程序的服务的“绑定”按钮。

(5) 绑定到服务之后必须再现应用程序。选择“再现>>”按钮可再现应用程序。

3. 访问应用程序中的服务

一旦应用程序绑定到服务，便会退出该应用程序（或重新启动），服务实例的凭据会在名为 VCAP_SERVICES 的环境变量中交付到应用程序运行时。VCAP_SERVICES 返回一个 JSON 文档，该文档包含允许应用程序访问服务的对象。您可以通过调用以下命令来访问应用程序的环境变量：

Cf env APP-NAME

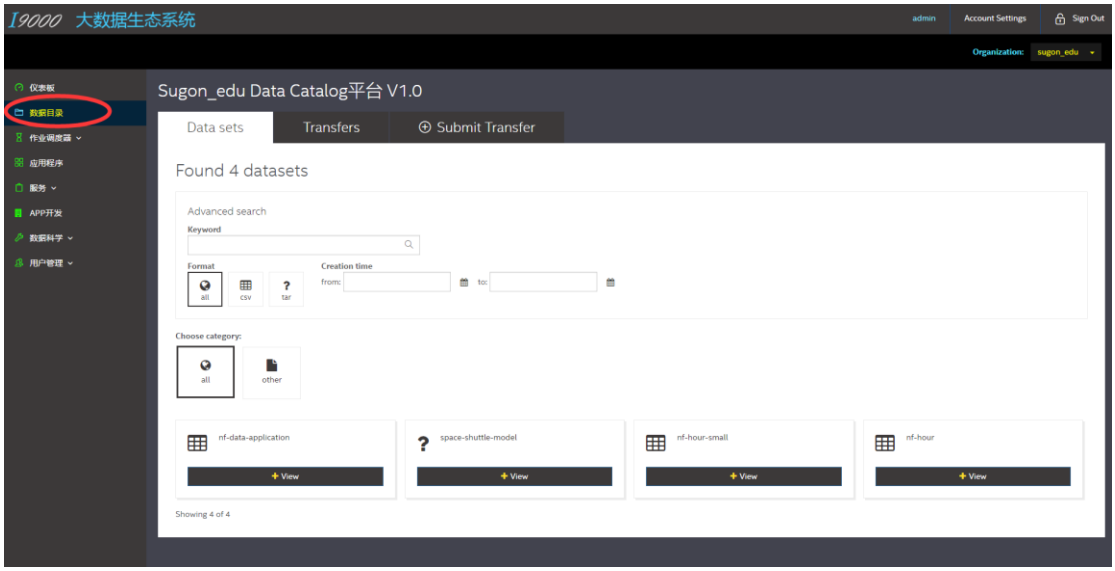
有关详细信息，请阅读 Cloud Foundry 文档。

四、I9000 中的数据获取

1. 数据获取

1.1 将数据导入 I9000

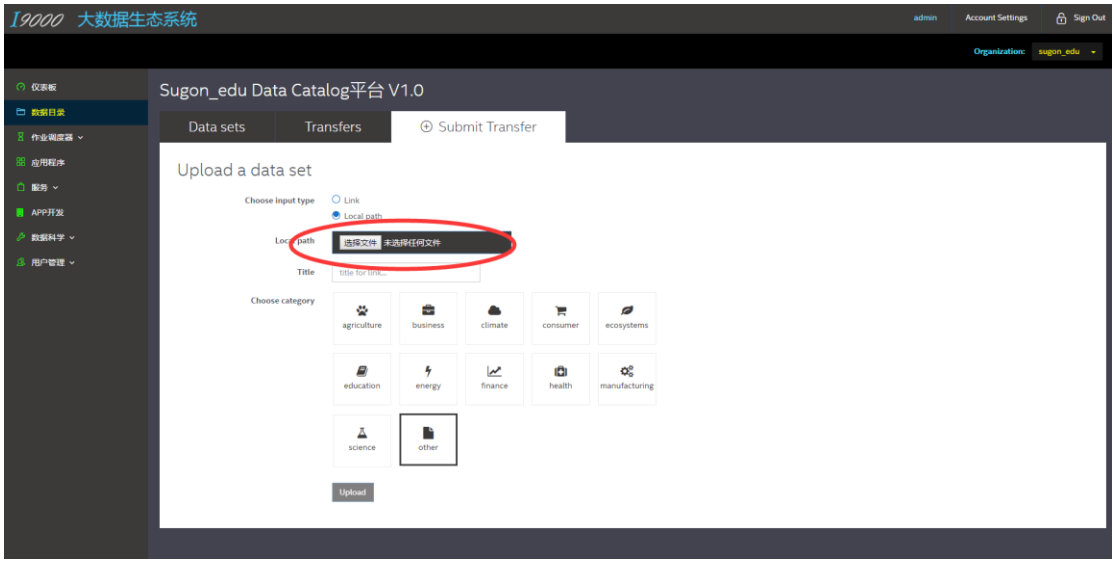
(1) 从“>主页”的主菜单中选择“>数据目录”



图片 31 数据目录

(2) 选择“>提交转移”选项卡

(3) 选择“>链接”以输入数据集的 URL 或选择“>本地路径”以激活“选择文件”文件浏览器按钮，该按钮将打开您的本地文件目录浏览器。在选择文件或提供 URL 之前不会激活“上传”按钮。



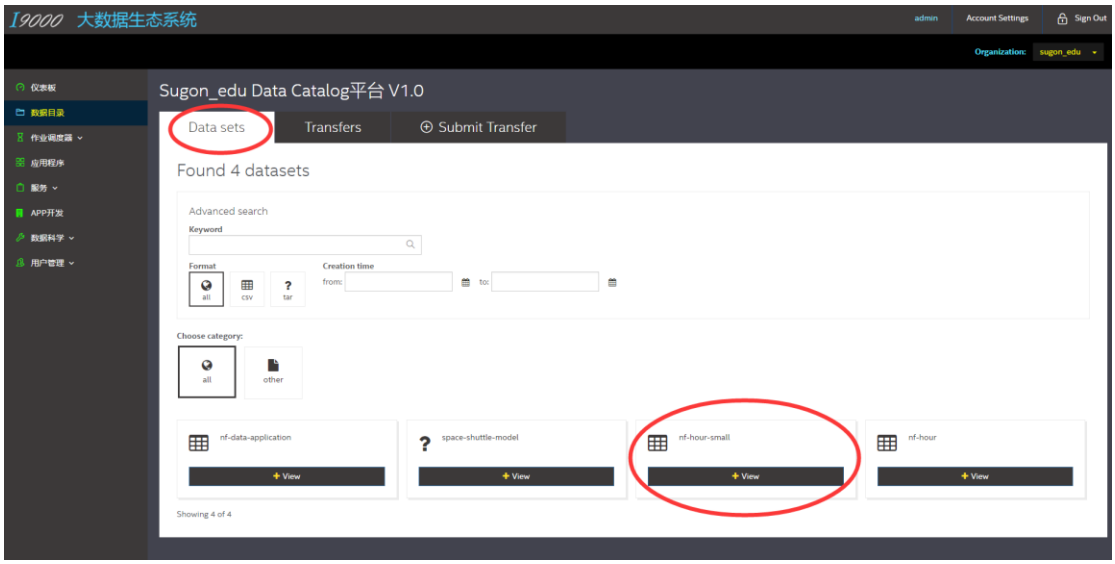
图片 32 上传本地文件

(4) 在标签为“标题”的表单字段中输入标题。在提供标题之前不会激活“上传”按钮。

(5) 选择一个类别以帮助其他人轻松找到您的数据。

(6) 选择“上传”按钮。

此时您的数据集将在“>数据目录”中列出。



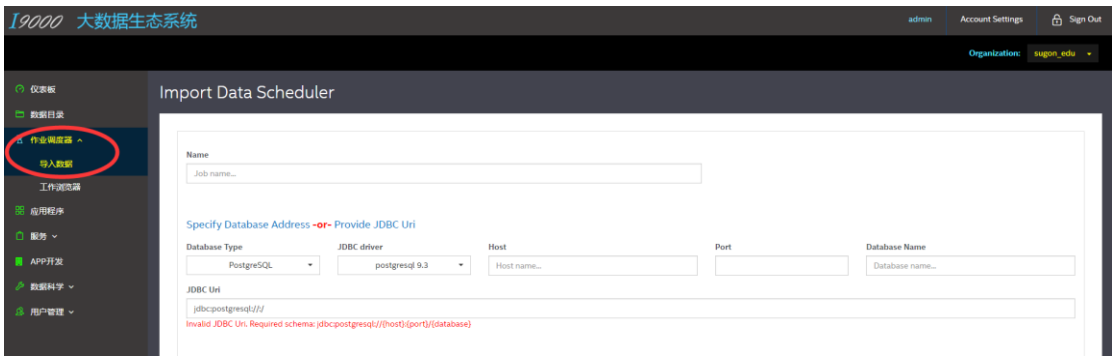
图片 33 查看数据文件

2. 从 SQL 源导入数据

作业调度器允许您将 SQL 数据库中的数据导入到 I9000 的 HDFS 中。可以采用批量模式导入数据，也可以定时导入数据或者自动更新。

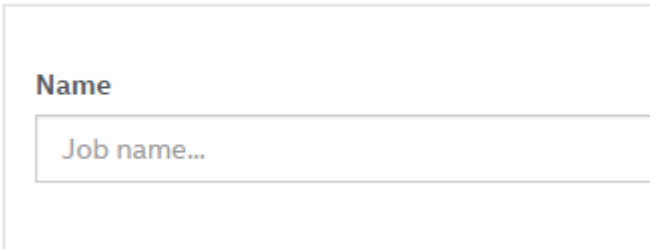
2.1 从 SQL 数据库导入数据

(1) 从 I9000 控制台主菜单中，导航到“作业调度器”，然后导航到“导入数据”。



图片 34 作业调度器

I9000 会显示一个供您填写的表单，从作业名称开始，如下所示。为您的作业选取一个独特的名称。



图片 35 作业名称

(2) JDBC Uri：您可以直接输入 URI，也可以填写 URI 字段上面的 Database Type 字段，以便为 JDBC URI 创建提供参数：

例如，您可以添加如下的 JDBC Uri 参数，以便启用 postgresql 连接的 SSL：

jdbc:postgresql://host:port/database_name?ssl=true&sslfactory=org.postgresql.ssl.NonValidatingFactory

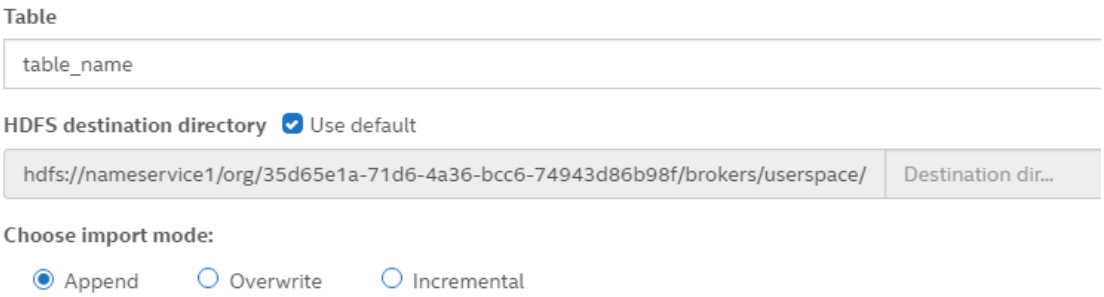
（请注意，参数是特定于驱动程序的，有关详细信息，请自行查看数据库相关文档）

图片 36 填写数据库信息

(3) 用户名和密码，这些是连接到数据源的凭据

图片 37 填写用户名和密码

(4) 数据库表，这是要导入到 HDFS 中的数据库表格的名称。



图片 38 填写表信息

(5) 目标目录，这是您将存储导入数据的目标 HDFS 的目录。

注意：确保您对该目录拥有写入访问权限。

(6) 选择导入模式，有 3 种导入模式可供选择：追加、覆盖和增量

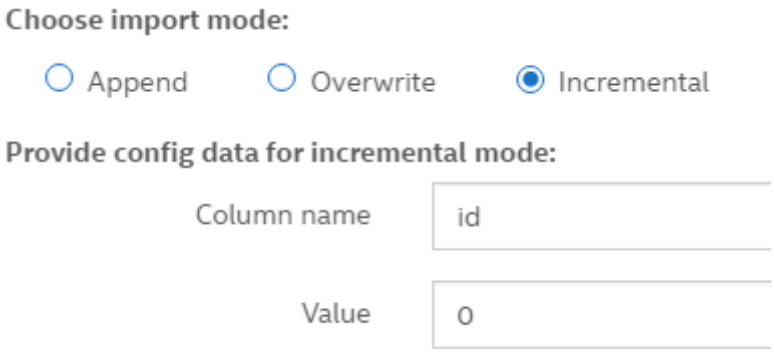
追加 – 每次导入都会将整个表格获取到一个单独的文件中。不会覆盖之前导入的结果。HDFS 上的文件将按照以下模式命名：part-m-00000、part-m-00001 以此类推。

覆盖 – 每次导入都会获取整个源表格并覆盖之前导入的结果，使用 part-m-00000 作为文件名。

增量 – 导入将获取在“列名称”参数标识的列中拥有不低于“值”参数所提供的值的记录。之后的每次导入都会获取先前提到的列中的值高于之前获取的值的记录。为此目的（标识），我们建议使用自动递增的数值列。

(7) 列名称：数据库中的列（唯一的数值格式），会根据该格式对值进行检查；用于唯一标识要导入的数据。

(8) 值：用于从源数据库中过滤记录的参考值。将只导入值（在“列名称”标识的列中）不小于该参考“值”的记录。



图片 39 填写列信息

(9) 开始时间，作业的开始时间。

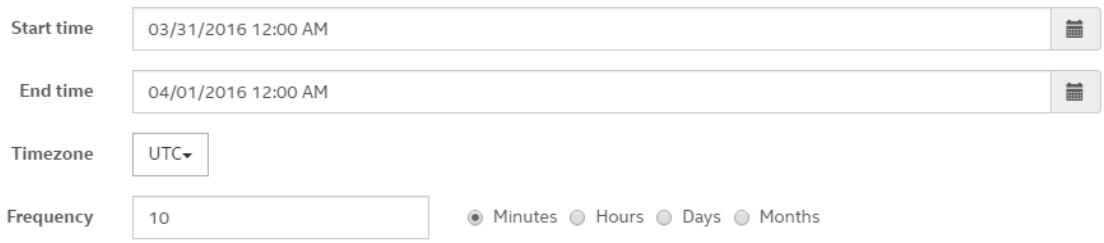
注意：当您输入一个在当前时间之前的某个“开始时间”时，Oozie 会尝试通过执行过去的作业来“追赶”。

(10) 结束时间，作业的结束时间。

“结束时间”应始终晚于“开始时间”。

(11) 频率，将提交作业的频率。

(12) 时区，输入的开始时间和结束时间的时区 ID。



图片 40 填写时区

2.2 工作浏览器

选择“作业调度器”，然后从 I9000 主菜单中选择“工作浏览器”可查看计划的作业。

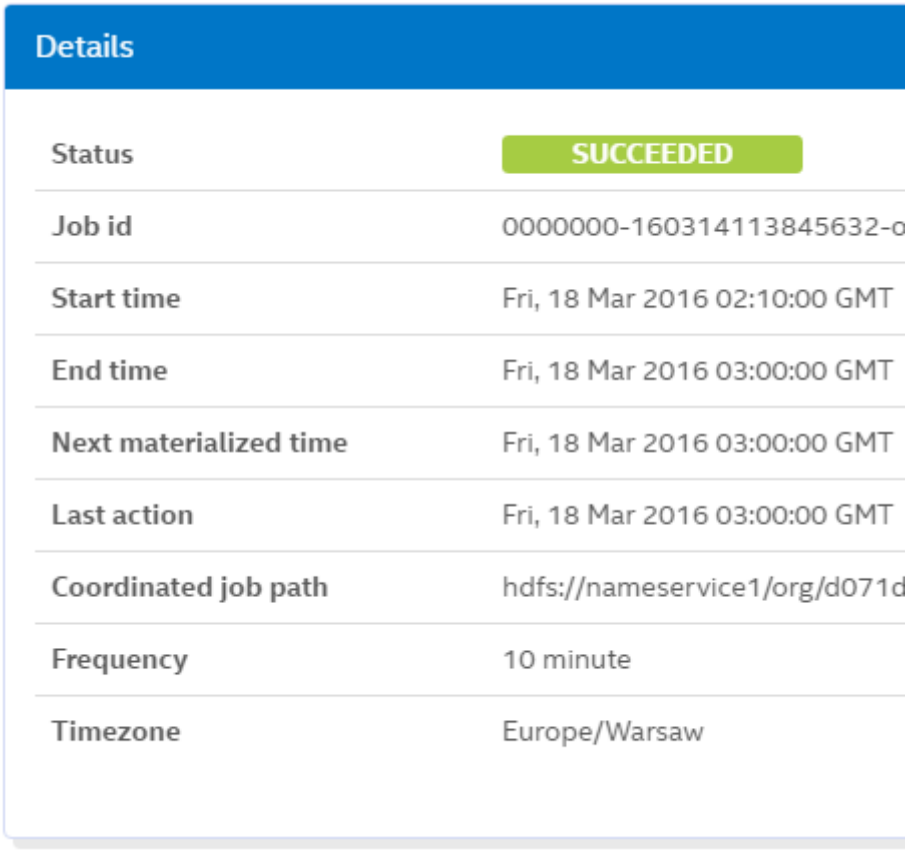
“工作浏览器”页面上有两个选项卡：“工作流作业”和“协调者作业”。

2.2.1 工作流作业

在该选项卡上，您可以看到工作流作业的列表。工作流作业表示从数据库到 HDFS 的导入。单击作业名称右侧的“查看详情”可获取详细信息（如下面的示例所示）。

详情：该部分提供有关指定工作流作业的详细信息。

查看日志 :该部分提供与指定工作流作业有关的日志。您可以通过单击“终止”按钮来终止作业。



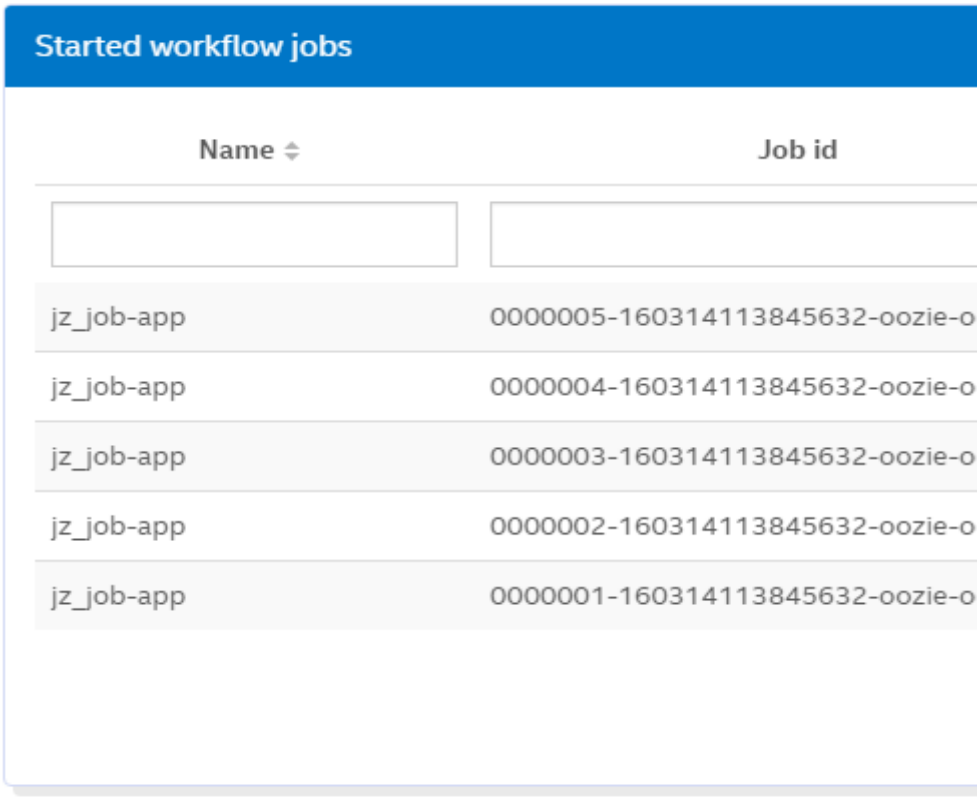
图片 41 查看工作日志

2.2.2 协调者作业

该选项卡包含配置信息并且可管理工作流作业。单击作业右侧的“查看详情”可获取详细信息（如下面的示例所示）。

详情：有关协调者作业的详细信息。

已开始的工作流作业：由协调者作业生成的工作流作业列表。该列表上的每个工作流作业都有一个“查看详情”链接，该链接会将您重定向到所选择的工作流作业详情。



图片 42 查看工作列表

五、I9000 分析工具包(ATK)

1. ATK 概述

ATK 提供一个可扩展 API，可用于 ETL、特征工程、图形构建和查询、ML 分析与深度编程语言集成，它包含如下几个部分：

数据科学家的 Python 包：使大数据更容易使用。

REST Web 服务：保护客户端，了解正在使用的大数据服务的详细信息。

正在处理的引擎：协调、执行、监视、审计，实现所有分析和机器学习。

1.1 API 功能

对表格数据的操作：

从 HDFS 文件导入；

添加/删除列、填补缺失值；

过滤、加入；

机器学习。

对图形数据的操作：

从表格数据框加载；

查询；

机器学习。

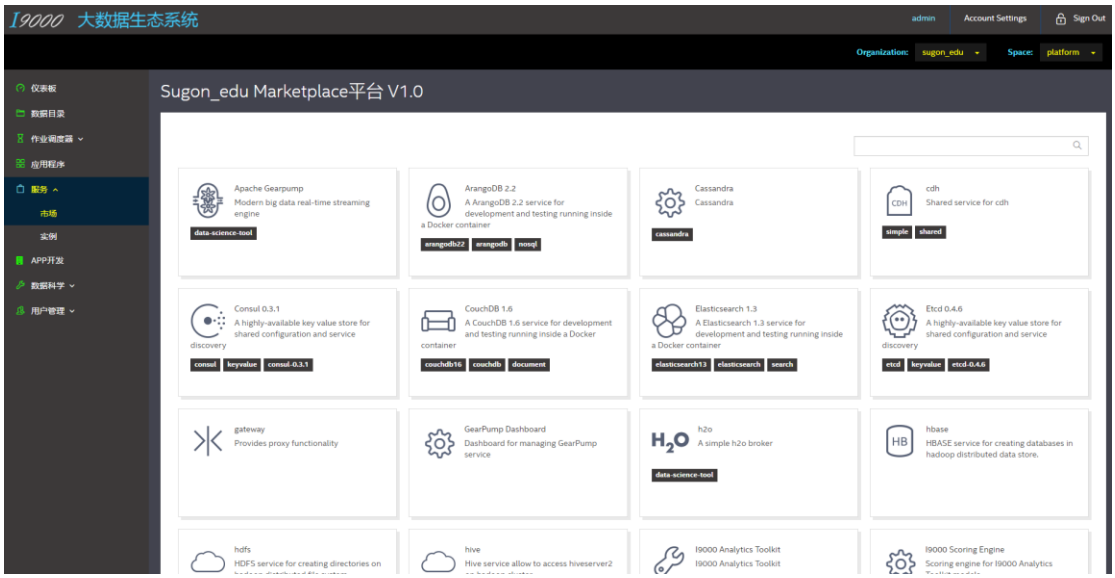
2. 如何创建分析工具包的新实例（两种方法！）

在 I9000 中可以采用两种方法来创建分析工具包的新实例。

2.1 通过服务

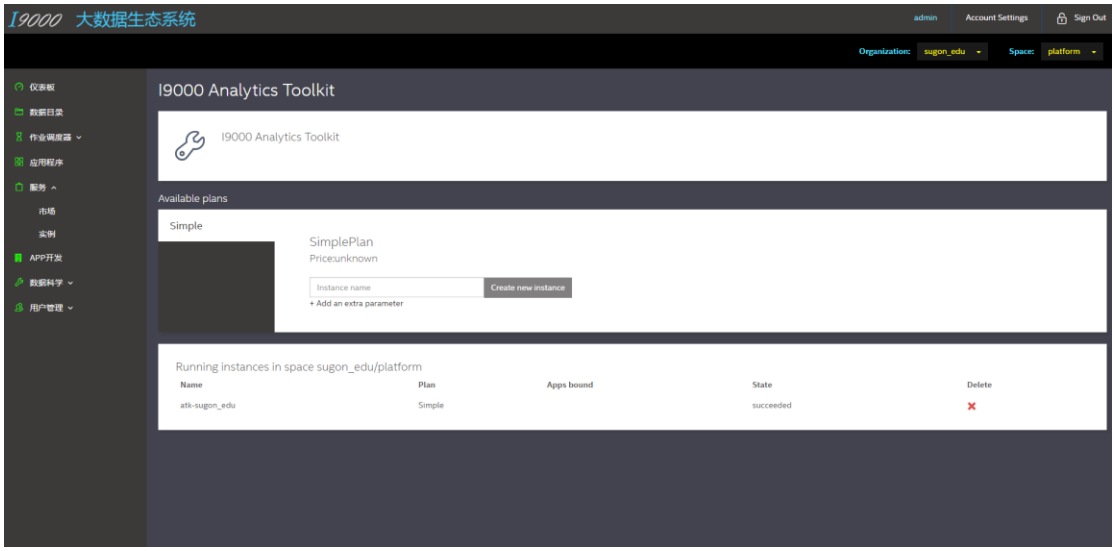
单击左侧导航上的“服务”。将出现两个选项，其中包括“市场”和“实例”。

单击“市场”。将出现一个服务目录。搜索“I9000 分析工具包”或滚动。单击“I9000 分析工具包”。您可以看到一个已安装的实例列表。



图片 43 市场

键入“实例名称”并单击“创建新实例”。可能需要几秒钟才能创建该服务，并且会出现通知。



图片 44 创建 ATK 实例

重新加载该页面即可看到新实例已列出。

2.2 通过数据科学

单击左侧导航上的“数据科学”。将出现几个选项，其中包括“I9000 分析工具包”。

单击“I9000 分析工具包”。您可以看到一个已安装的实例列表。

键入“实例名称”并单击“创建新实例”。可能需要几秒钟才能创建该服务，并且会出现通知。

重新加载该页面即可看到新实例已列出。

3. 使用 ATK

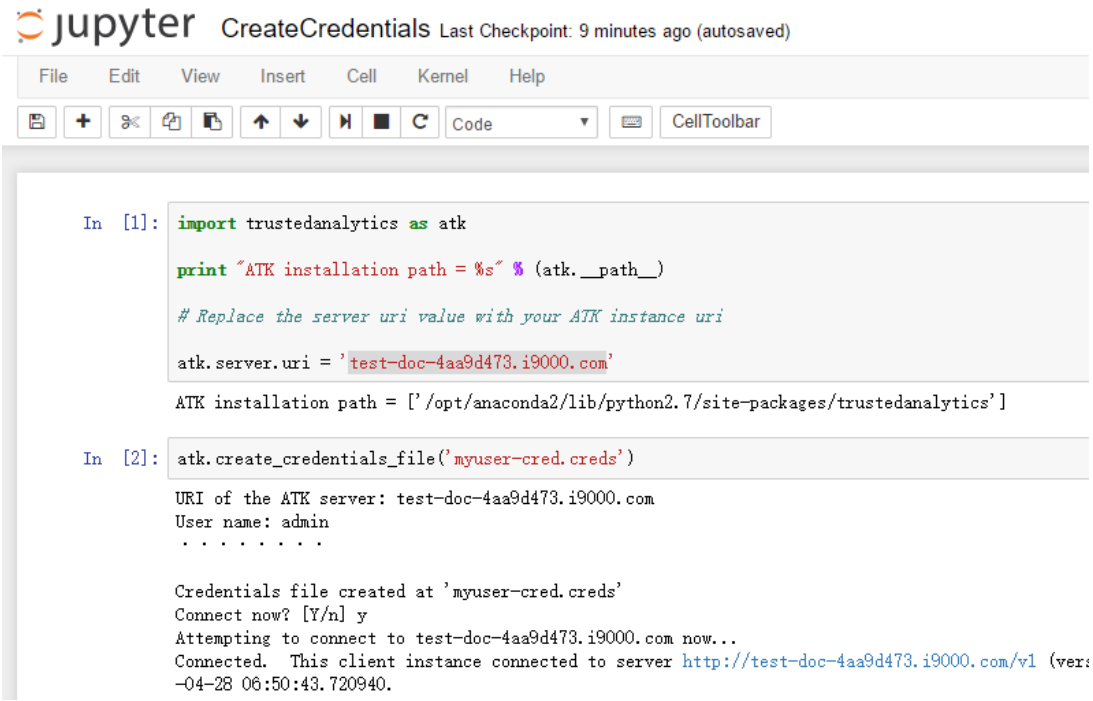
若要使用 I9000 分析工具包(ATK)，您将需要创建一个新的 Jupyter Notebook 实例并且需要创建用于连接到 ATK 的一组凭据。以下步骤将指导您完成此过程。

(1) 创建一个 Jupyter Notebook 实例。

(2) 创建您的用户凭据文件：仅当您第一次尝试使用分析工具包或切换到分析工具包的一个新实例时才需要采取此步骤。

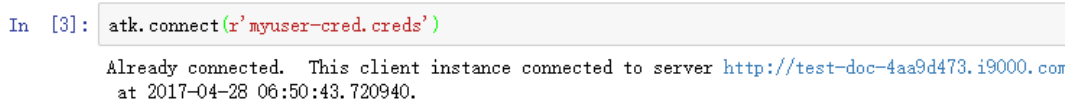
(3) 创建一个名为 CreateCredentials 的 ipynb 文件，输入如下所示脚本，并运行脚本。

注意：create_credentials_file 函数会要求您输入 ATK 的 URL 和您的用户名与密码。



图片 45 创建 ATK 凭据文件

(3) 创建凭据会生成一个名为 myuser-cred.creds 的文件。当编写从 Jupyter Notebooks 连接到分析工具包的脚本时需要使用该文件。详细参照下图



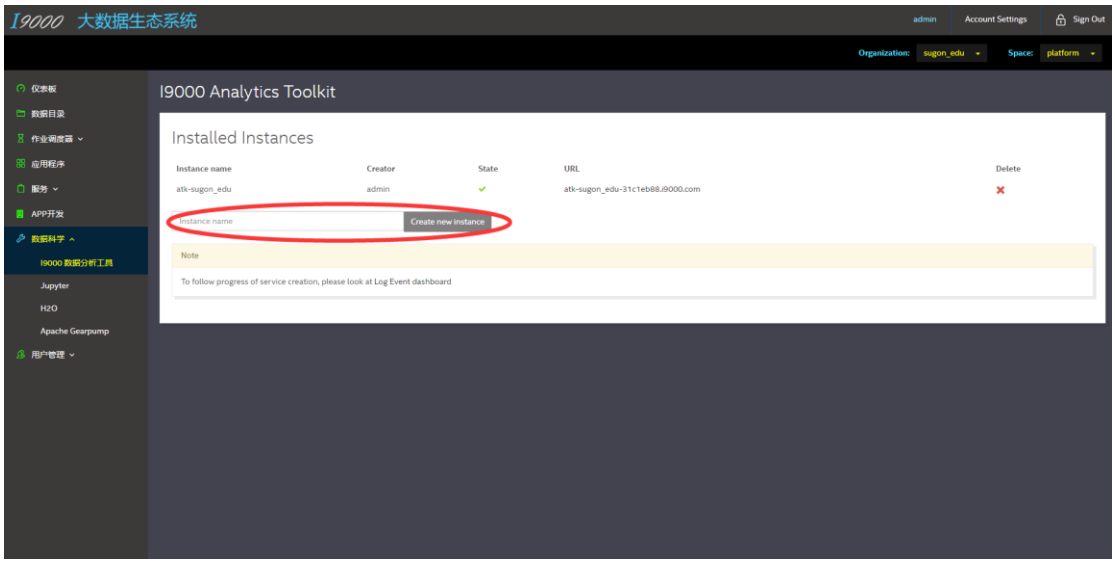
图片 46 测试凭据文件

4. 客户端安装配置

您也可以选择在本地安装 ATK 客户端，来更方便的使用 ATK 工具包。

4.1 创建工具包服务器实例

您可以参考本章第 2 节所提供的方式创建一个 I9000 Analytics Toolkit 软件服务器。例如直接在“数据科学”键入所需的工具包服务器实例名称并单击“创建新实例”按钮。



图片 47 创建服务实例

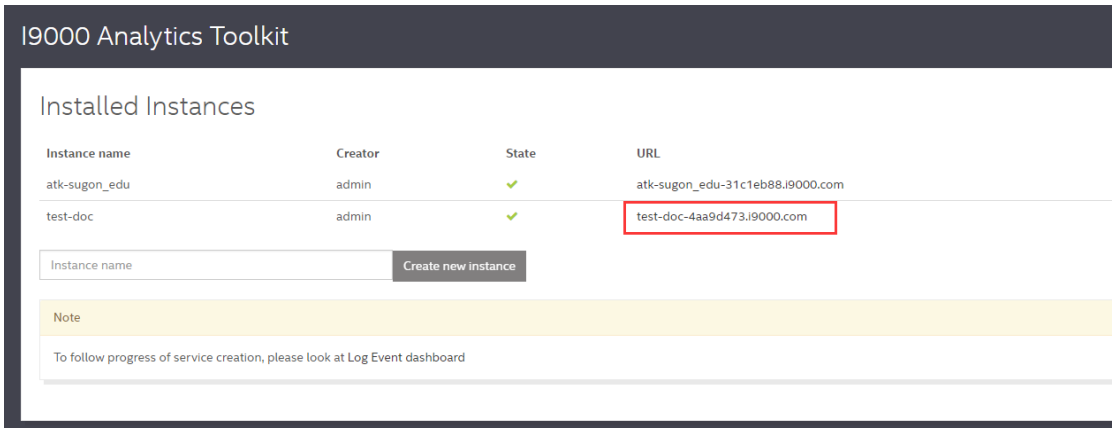
4.2 Windows 上的安装

(1) 若要在基于 Windows 的计算机上使用 ATK 客户端，您将需要安装 Anaconda Python 分发包。您可以通过位于 Anaconda 下载页面上的 msi 来安装该软件。

(2) 安装 ATK 客户端之前，您必须首先通过 Anaconda 安装 NumPy 和 Pandas 组件。

(3) 打开“数据科学”中的 I9000 数据分析工具，查看刚刚创建的 ATK，点击后面的 URL

获取对应的 ATK 客户端安装指令（详见如下两个截图）。



图片 48 查看 ATK 的 URL 信息

图片 49 查看 ATK 客户端安装指令

(4) 打开命令行终端，使用上图所示的 pip 指令来安装 ATK 客户端。

4.3 Linux 上的安装

(1) 安装工具包客户端之前，您将需要安装 Python 2.7 和 pip。

(2) 安装 Python 2.7 和 pip 之后，按照 windows 上安装的方式安装 ATK 客户端。

4.4 配置

安装 ATK 客户端之后，您必须将其配置改为连接到所需的 REST 服务器。

六、应用程序开发和部署

1. 开发环境设置

1.1 操作系统

安装 Ubuntu 操作系统（版本 14.04 或更高版本）。

注意：如果系统上未安装该文档中所列出的安装其他程序包所需的任何程序包，则会通知您控制输出中缺少所需的程序包。可以使用以下命令来安装这些程序包：

sudo apt-get install <package_name>

1.2 Cloud Foundry 命令行

导航到 I9000 平台的“APP 开发”，在此页面根据您的系统，选择不同版本的 CLI 客户端下载并安装，安装成功后，您就可以使用 Cloud Foundry 相关命令了。

1.3 开发人员

1.3.1 Java

添加以下存储库：

sudo add-apt-repository ppa:chris-lea/node.js

sudo add-apt-repository ppa:webupd8team/java

注意：如果控制台输出在尝试添加存储库时显示“sudo: add-apt-repository: command not found”，则使用此命令安装以下程序包：

sudo apt-get install software-properties-common python-software-properties

从存储库更新程序包列表：

sudo apt-get update

注意：如果您在添加 ppa:webupd8team/java 之后未更新程序包列表，那么您将无法安装 Java 8（“Unable to locate package oracle-java8-installer”）。

使用这些命令安装以下程序包：

sudo apt-get install oracle-java8-installer oracle-java8-set-default

1.3.2 Maven

如果您是 Maven 的新手，请在下面网页上了解详细信息：https://maven.apache.org/guides/getting-started/

sudo apt-get install maven

1.3.3 Node.js

sudo apt-get install nodejs

1.3.4 Python

sudo apt-get install python python-dev python-setuptools python-pip

Install the following packages using pip:

sudo pip install virtualenv

sudo pip install tox

1.3.5 Git

sudo apt-get install git

1.3.6 外部函数接口库

sudo apt-get install libffi-dev

1.3.7 Go

(1) 默认情况下安装 Go 版本 1.6：

wget https://storage.googleapis.com/golang/go1.6.linux-amd64.tar.gz

mv go/* ~/go1.6

rm -Rf go

export GOROOT=`echo ~/go1.6`

export PATH=$GOROOT/bin:$PATH

注意：如果您想永久导出这些变量，则将这些导出添加到您的 ~/.bashrc 文件中。

(2) 如果您遇到编译错误，则应将您的 git 客户端更新到最新版本。

(3) 安装 gccgo-go

sudo apt-get install gccgo-go

(4) 设置 Go 工作区：

mkdir ~/go-workspace

export GOPATH=$HOME/go-workspace

export PATH=$PATH:$GOPATH/bin

注意：同样，如果您想永久导出这些变量，则将上面这两个导出添加到您的 ~/.bashrc 文件中。

(5) 安装 Godep

go get github.com/tools/godep

1.3.8 Ruby

(1) 下载 GPG 签名：

gpg –keyserver hkp://keys.gnupg.net –recv-keys

409B6B1796C275462A1703113804BB82D39DC0E3

(2) 安装 RVM (Ruby Version Manager)：

curl -L https://get.rvm.io | bash -s stable

(3) 执行：

source ~/.rvm/scripts/rvm

(4) 默认情况下安装 Ruby 版本 2.1.5 并使用它：

rvm install ruby-2.1.5

rvm use 2.1.5

(5) 您可能还想生成 Ruby 文档：

rvm docs generate-ri

(6) 安装 gems：

gem install compass

gem install cf-uaac

1.4 区域设置

为了避免构建平台组件时出现问题，请确保您的区域设置使用 UTF-8 编码（有关检查和更改区域设置的信息，请参阅 https://help.ubuntu.com/community/Locale）。

您的系统现在已准备好复制您选择的存储库并且可以开始进行开发工作了。

2. 开发指南

2.1 编码风格

2.1.1 Java

使用从此处导入的编码风格：google-styleguide。导入最新版本的 Eclipse 时您可能会遇到问题。您可能需要手动将缩进长度设置为四个空格。该文件非常适合用于已安装该插件的 IntelliJ Idea。

2.1.2 Python

遵循标准的 PEP8 风格。默认情况下 Pycharm（它基本上是带有 Python 插件的 Idea IDE）检查 PEP8 遵从性。PEP8 建议您应该做的唯一不同的事项是文档字符串。使用 reStructuredText 风格格式化文档字符串，如下所示：

```
class ExampleClass():
    """
    This docstring says about the role of this class.
    """
    def example_method(self, count):
        """
        This method creates a list of consecutive numbers of the given length.
        :paramint count: Length of the requested list.
        :returns: A list of consecutive numbers, starting from 0.
        :rtype: list[int]
        :raises ValueError: When count is less than zero.
        """
        if count < 0:
            raise ValueError('count can't be lesser than zero')
        return [i for i in range(count)]
```

注意：记得记录每个公共方法！

PyCharm 为这些类型的文档字符串生成存根。在函数或类声明之后键入 """ 或 ''' 并

按 Enter 或空格键。

单元测试

根据以下内容对每个测试的状态使用一个单元测试：(http://blog.8thlight.com/uncle-bob/2013/09/23/Test-first.html)

使用以下命名惯例：testedFunctionName_testedState_expectedResult 或 checkToken_tokenInvalid_returnFalse 。在以下博客上了解有关此概念的详细信息：(http://osherove.com/blog/2005/4/3/naming-standards-for-unit-tests.html)

为了编写出色的 assert 消息，请使用有意义的 assert 而不是“assertThat(result, is(expected))”。

不要使用字段注入，而是注入到构造函数中，无需模拟私有字段等。

2.1.3 License Maven Plugin 和 Pre-commit Git Hooks

License Maven Plugin 是一个命令行工具，它允许我们更改源文件中许可证标题的样式和内容。

如果您使用的是 Java 项目，则应在根目录中有一个 pom.xml 文件。这种情况下，您只需从终端使用该工具：

mvnlicense:check	检查所有源文件是否都有正确的标题
mvnlicense:remove	删除标题
mvnlicense:format	添加标题

如果您未使用 Java 项目（bash、go、scala...），则应在 license_checker 目录中有

一个 license_checker.xml 文件。在这种情况下使用：

mvn -f license_checker/license_checker.xml license:remove

您可以添加该功能以在提交之前检查您的文件是否具有正确的标题。您可以通过采取以下步骤来完成该操作：

open project_root_directory/.git/hooks

找到一个 pre-commit.sample 文件并将其重新命名为 pre-commit

对于 Java 项目，将内容更改为：

```
#!/bin/sh
cd license
./header_check.sh
```

对于其他语言，将 cd/license 更改为 cd/license_checker

3. 部署第一个应用程序工具

除了有一个用于编写应用程序代码的 IDE 之外，部署应用程序的主要工具是 CLI (Command Line Interface) for Cloud Foundry。您可以参考第 1 节内容。

3.1 示例应用程序

“Spring Music”是一个非常有用的 Cloud Foundry 应用程序，您可以从此处的 github 复制该应用程序：

https://github.com/cloudfoundry-samples/spring-music

该应用程序需要对该平台有一些了解，并且还采用 MongoDB 的挂接（这是需要熟悉一个出色的服务）。

我们还构建了一个示例应用程序以便用于该平台以及从 hdfs 拉取数据。例如：dataset-reader-sample。我们围绕该应用程序创建了一套完整的工作流程，详细介绍了如何围绕获取的数据在 I9000 中构建应用程序。

3.2 组装

为了将 Java 应用程序部署到大数据分析平台，您必须使用一个工具（如 Maven）来组装它们。您可以在此处下载 Maven：https://maven.apache.org/download.cgi。如果您是 Maven 的新手，下面是更好地了解构建/组装过程的步骤：https://maven.apache.org/guides/getting-started/。

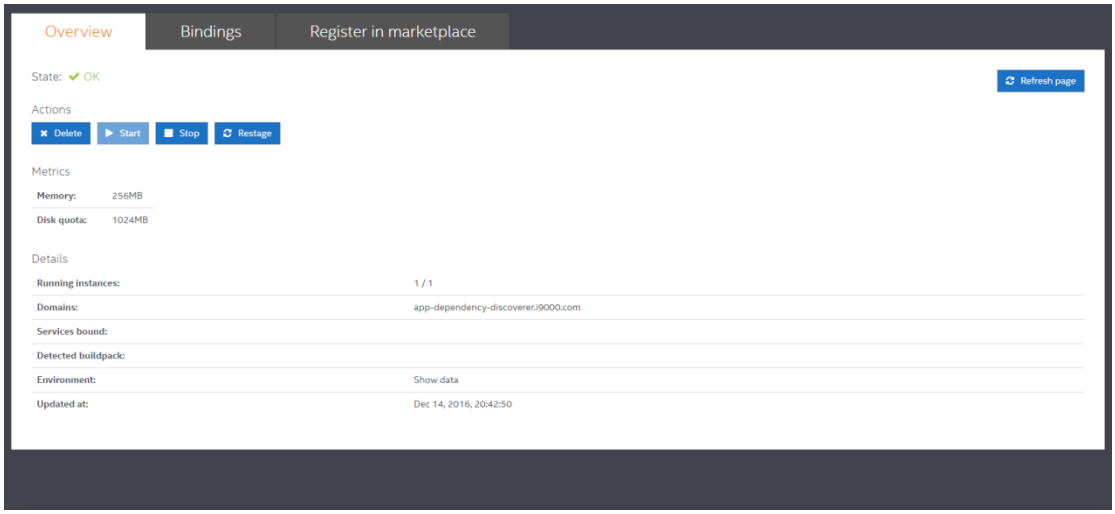
3.3 推送应用程序

使用 Cloud Foundry CLI 将应用程序上传到 I9000 实例，以下是一些 CLI 的相关命令：

```
$ cf login -a <Your I9000 instance Endpoint>
  API endpoint: <Your I9000 instance Endpoint>
  Username>your_username
  Password>your_password
  Org>your_org
your_space
$ cf push your_app
```

3.4 在 I9000 中查看应用程序及其实例

成功上传应用程序之后，您便可以在 I9000 的实例上查看。转到“I9000 控制台”>>“应用程序”即可查看应用程序的实例。单击 URL 将加载该应用程序。单击“查看详情”会显示该应用程序的配置文件以及用于启动、停止、再现该应用程序的控件。



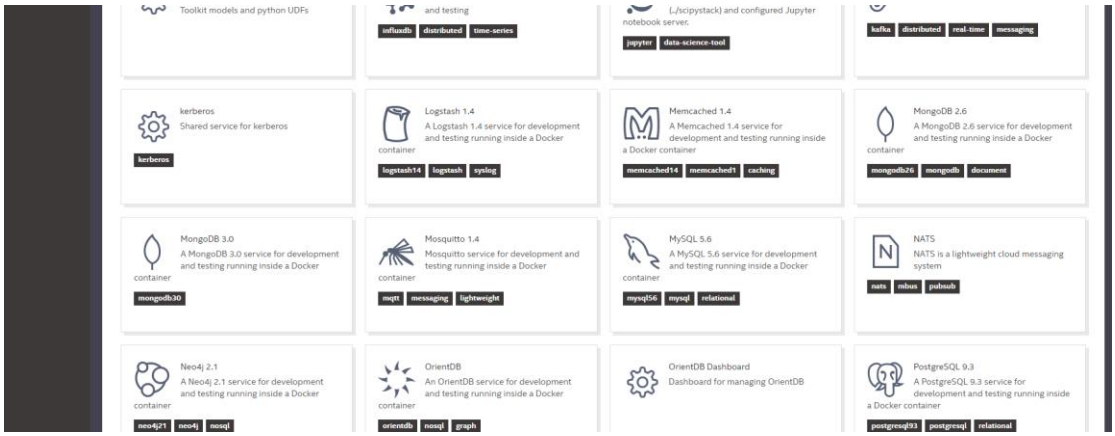
图片 50 查看应用程序

3.5 绑定服务

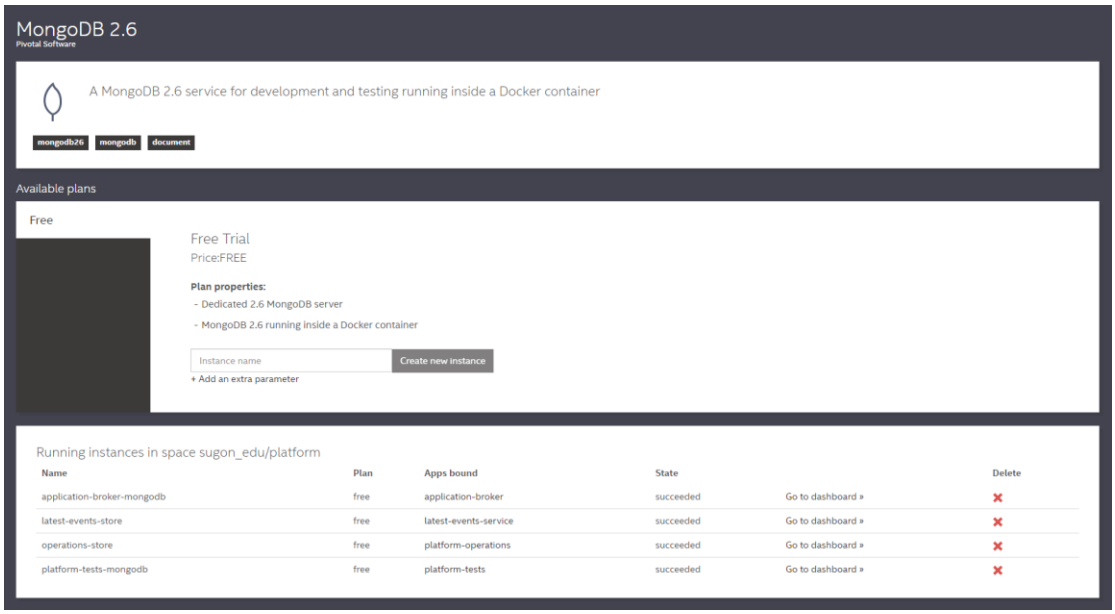
使用“控制台”>>“市场”中的 I9000 的服务代理可以向应用程序中轻松添加流行的服务，如 MongoDB。I9000 和 Cloud Foundry 的这一突出的可扩展方面可以轻松地将数据存储的实例以及第三方服务集成到您的应用。也可以通过编程方式添加服务，包括可让您自己的服务器日志用作服务。

继续使用上面的“Spring Music”应用程序示例，您可以轻松地创建 MongoDB 服务的实例并将其绑定到您的应用程序。这是非常有用的，因为到现在为止，应用程序只在自己的内存中存储条目。如果应用程序需要再现，它将失去它的所有数据。

转到“控制台”>>“市场”并浏览到“MongoDB”服务。单击该服务并在该服务的配置文件中添加一个名称为“myMongo”新实例。可用之后，该实例将列出哪些空间可用（参见下图）。



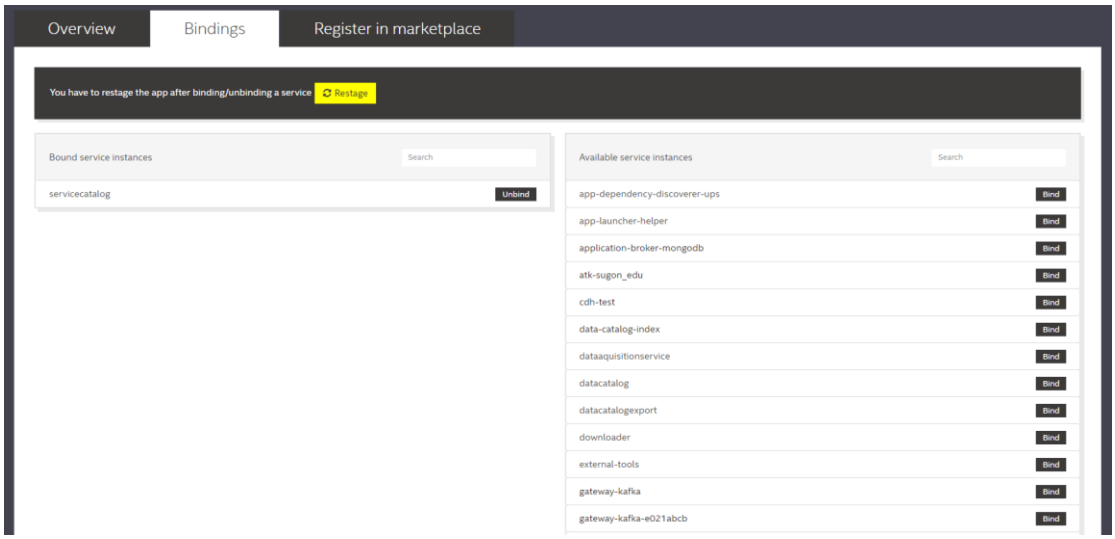
图片 51 市场



图片 52 创建 MongoDB 实例

回到“控制台”>>“应用程序”，通过单击“查看详情”链接查看您的“app-dependency-discoverer”配置文件。单击“绑定”选项卡。您应该会看到右侧的

“可用的服务实例”中列出了“myMongo”实例。单击“绑定”。现在会提示您单击“restage”以便再现该应用程序并绑定到您创建的 MongoDB 实例。再现之后，输入的所有数据都将存储在 MongoDB 实例中，从而为该应用程序创建永久的数据存储。



图片 53 绑定页面

4. Java 的技巧和窍门

4.1 通过 Spring 读取应用程序属性

您可以通过 Spring 轻松读取复杂的环境变量。例如：

```
"VCAP_APPLICATION": {
    "uris": [
        "sample.url1.com",
        "sample.url2.com"
    ]
```

```
}
```

检索 uris 阵列中的第一个 URI。在 application.yml 文件中，键入：

```
app:
    url: ${vcap.application.uris[0]:default}
```

您的 code-config 文件应如下所示：

```
@Configuration
public class Config {
    @Value("${app.url}")
    private String myAppUrl;
}
```

“sample.url1.com”将插入到 Config 类的 myAppUrl 变量中。

4.2 为 CF 应用程序启用 Java gc 日志

```
cf set-env app-name JAVA_OPTS "-verbose:gc -XX:+PrintGCDetails"
cf restage app-name
```

4.3 为 CF 应用程序指定堆和元空间大小

使用 Java buildpack 的最新版本（支持指定在 3.2 中添加的堆大小）：

```
cf push <APP-NAME> -p <ARTIFACT> -b
https://github.com/cloudfoundry/java-buildpack.git
```

设置环境变量：

cf set-env<APP-NAME> JBP_CONFIG_OPEN_JDK_JRE: [memory_calculator: {memory_sizes: { heap: 64M, metaspace: 34M }}]

再现应用程序。

cf restage app-name

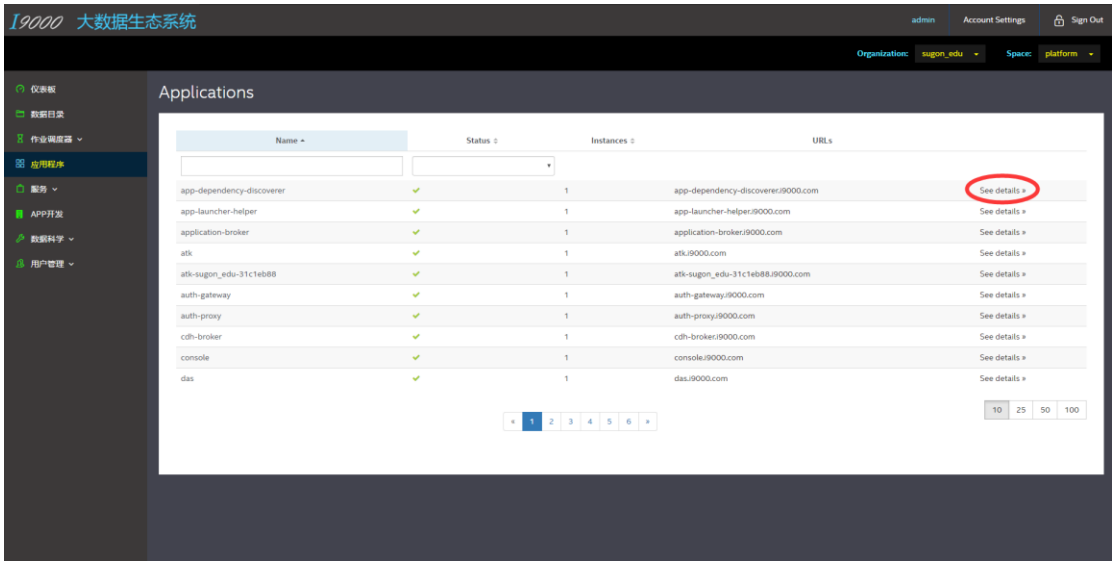
5. 应用程序代理

5.1 在市场中注册应用程序

应用程序代理是一个组件，它允许您将应用程序转换为 I9000 服务市场中的产品。您可以通过用户界面或命令行使您的应用程序在市场中可用。以下步骤将指导您完成这两个选项。

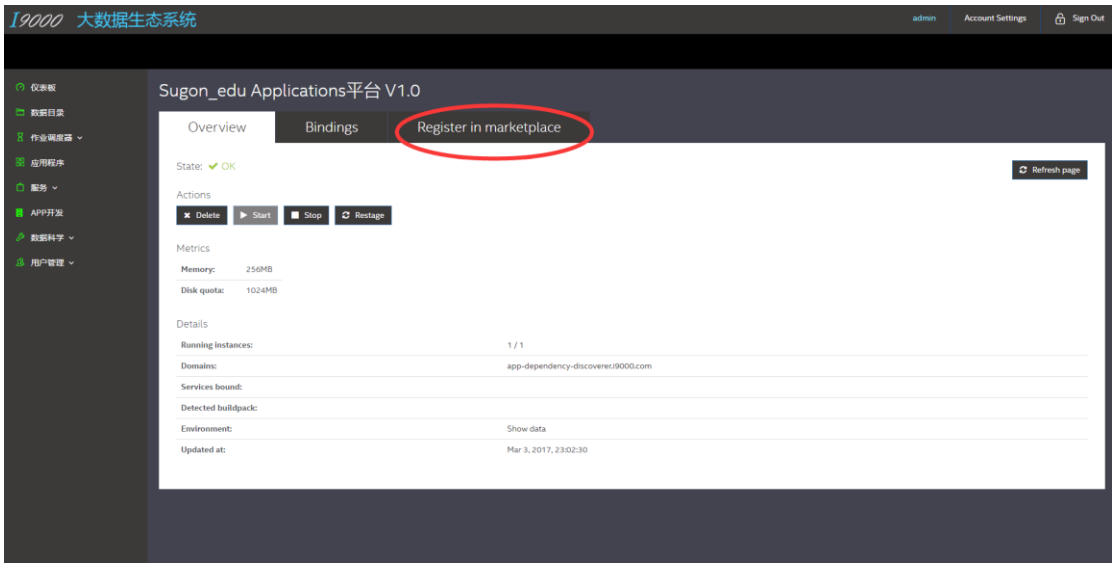
5.1.1 通过控制台用户界面

(1) 从“应用程序”页面中 ,选择您的应用程序行中的> “查看详情>>”链接即可访问应用程序概述页面。



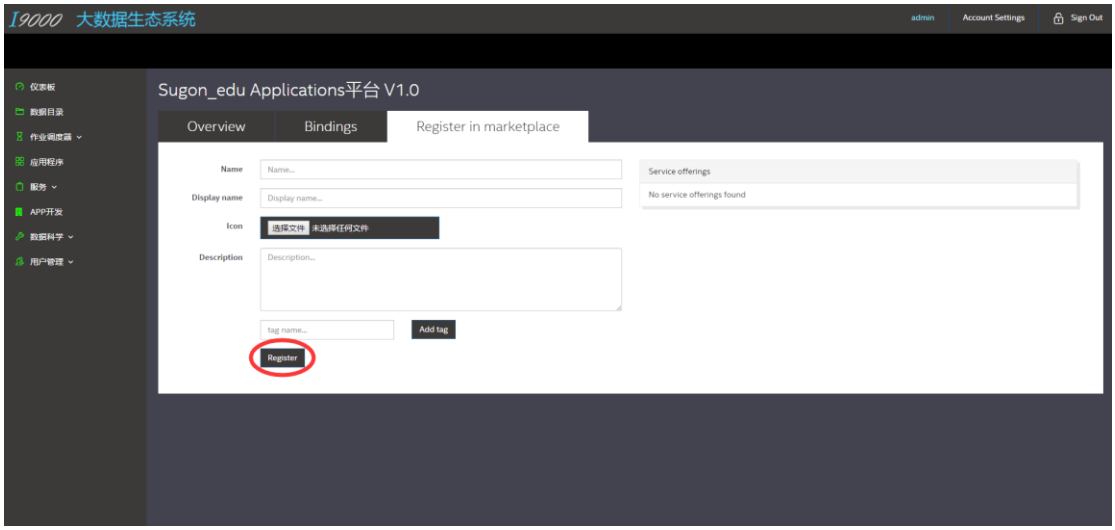
图片 54 查看应用程序详情

(2) 从应用程序概述页面中，选择>“在市场中注册”选项卡。



图片 56 注册到市场

(3) 填写提供的表单并选择>“注册”。



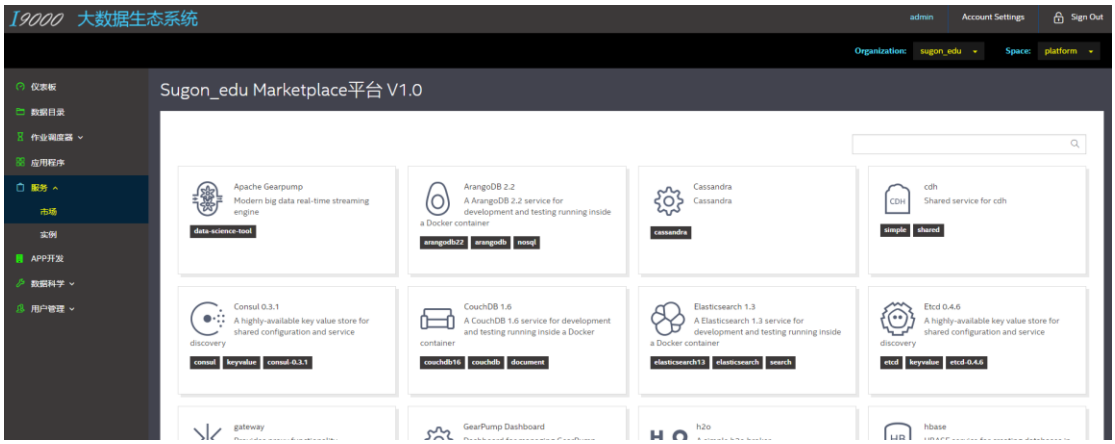
图片 57 服务注册页面

您的应用程序现在将位于>“服务”>“市场”中，可供您组织中的其他人使用。

5.2 从服务市场中删除已注册的应用程序

5.2.1 通过控制台用户界面

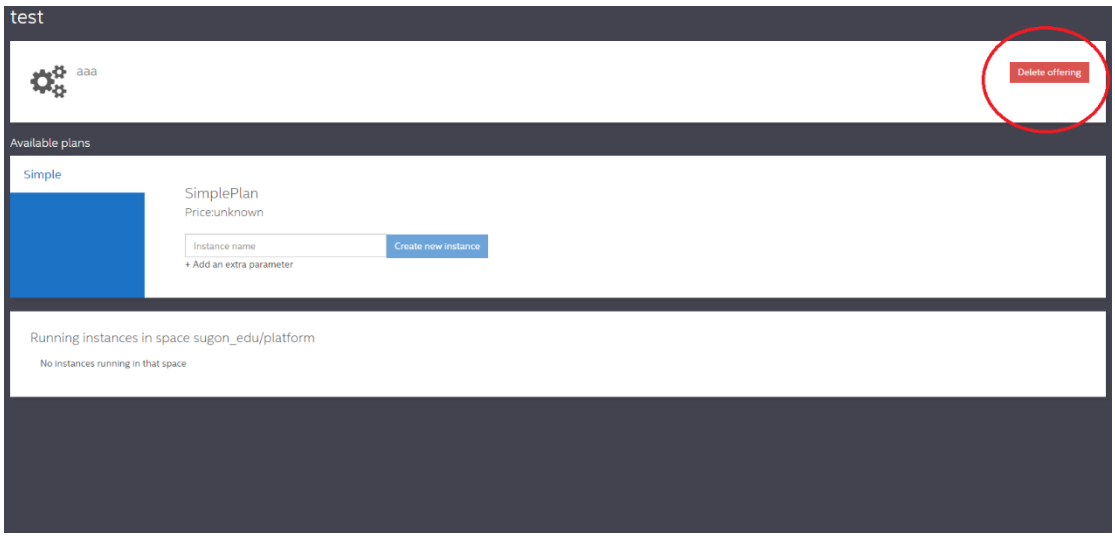
(1) 从主菜单中选择>“服务”>“市场”。



图片 58 市场

(2) 搜索并选择应用程序的名称以查看服务详情页面。

(3) 选择位于右上角的> “删除”按钮。



图片 59 删除服务

(4) 确认删除，现在该应用程序将已从“市场”中删除。

6. Kerberos 身份验证

Kerberos 代理提供 Kerberos 身份验证所需的基本配置。同样，对于 I9000 平台，您应该使用 Cloud Foundry 中的 JWT 令牌进行 Kerberos 身份验证。基于 JWT 令牌获取票证之后，您将以具有适当权限的 Cloud Foundry 用户身份在 Hadoop 站点上操作。

您可以通过以下方式确定 Hadoop 身份验证方法

property in Kerberos broker credentials (like below)

property in core-site.xml: hadoop.security.authentication=kerberos (or simple)

6.1 Kerberos 代理配置

创建并绑定 kerberos 类型的服务实例：

cf cskerberos shared kerberos-instance

cfbs<app>kerberos-instance

之后，您可以在应用程序环境中找到 Kerberos 的部分：

```
"VCAP_SERVICES": {
  "kerberos": [
   {
    "credentials": {
     "enabled": true,
     "kcacert": "<encoded cacert",
     "kdc": "cdh-manager-0.node.Bigdataanalytics.consul",
     "kpassword": "cf1",
     "krealm": "CLOUDERA",
     "kuser": "cf"
    },
    "label": "kerberos",
```

```
      "name": "kerberos-instance",
      "plan": "shared",
      "tags": [
        "kerberos"
      ]
    }
  ],
```

属性说明：

enabled	该标志确定身份验证方法，基于 Kerberos 还是 Simple(不使用 Kerberos)
kcacert	对 KDC pkinit 机制的证书进行了编码，以便使用 JWT 令牌获取 tgt 票证
kdc	密钥分发中心地址
krealm	Kerberos 领域
kuser	技术系统用户（不推荐使用）
kpassword	技术用户的密码（不推荐使用）

6.2 Java 项目的身份验证

Hadoop Utils 项目

Kerberos 登录管理器允许使用以下方法登录：

```
//Getting config properties values

String kdc = krbConf.getProperty(Property.KRB_KDC).get();

String realm = krbConf.getProperty(Property.KRB_REALM).get();

//Login in hadoop for authorized operating on HDFS

KrbLoginManagerloginManager =

KrbLoginManagerFactory.getInstance().getKrbLoginManagerInstance(kdc, realm);
```

推荐在 I9000 上使用 Kerberos 身份验证方法。使用令牌，您的应用程序将作为 Haddop 中的 Cloud Foundry 用户。

JWT Token: loginWithKeyTab(String user, String path)

```
String user = krbConf.getProperty(Property.USER).get();

loginManager.loginInHadoop(loginManager.loginWithKeyTab(user,
pathToFile),hdfsConf.asHadoopConfiguration());
```

仅适用于系统用户的方法：

Keytab file: loginWithJWTtoken(JwtTokenjwtToken)

```
I9000OauthToken jwtToken = new I9000OauthToken(tokenString)

loginManager.loginInHadoop(loginManager.loginWithJWTtoken(jwtToken
),hdfsConf.asHadoopConfiguration());
```

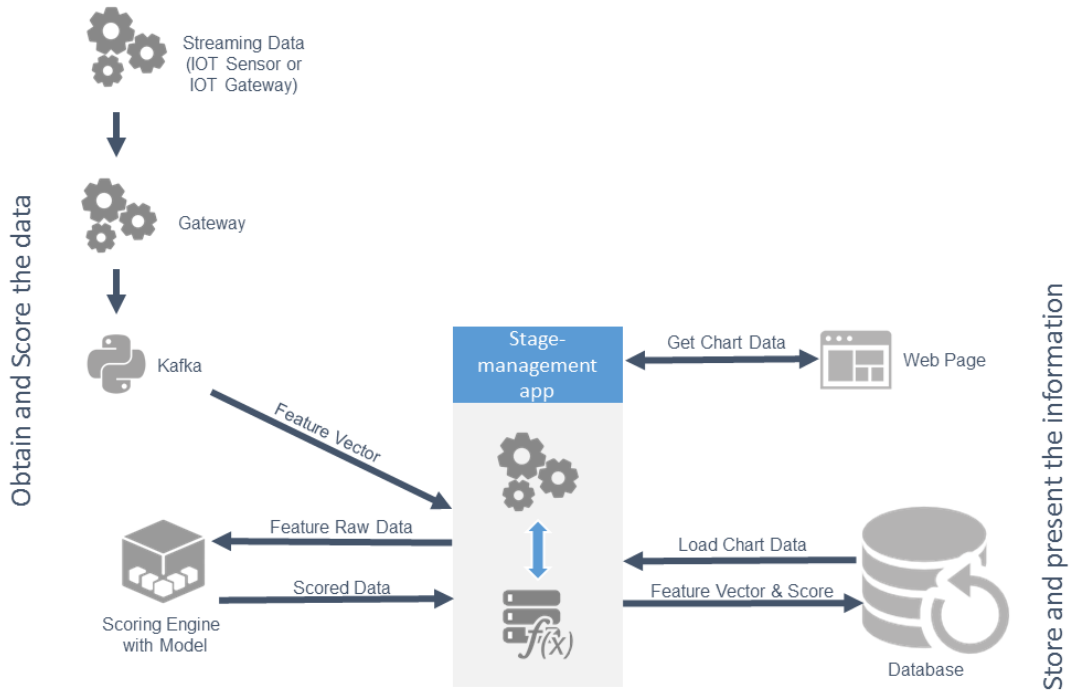
Credentials: loginWithCredentials(String user, char[] password)

```
String user = krbConf.getProperty(Property.USER).get();

String pass = krbConf.getProperty(Property.PASSWORD).get();

loginManager.loginInHadoop(loginManager.loginWithCredentials(user,
pass.toCharArray()),hdfsConf.asHadoopConfiguration());
```

七、示例应用程序

1. 航天飞机演示应用程序

1.1 概述



图片 60 航天飞机案例图示

1.1.1 实施概要

评分流程：

space-shuttle-demo（航天飞机演示）应用程序监听 Kafka 主题并等待特征向量。当出现 Kafka 消息时，该应用程序请求评分引擎对收到的特征向量进行分类。该应用程序将评分结果存储在 InfluxDB 数据库中。

生成图流程：

该 Web 应用程序要求后端应用程序（space-shuttle-demo）获取异常图表。space-shuttle-demo（航天飞机演示）应用程序每分钟从 InfluxDB 获取几次异常，然后显示这些异常。

1.2 将应用程序部署到 I9000

(1) 将模型 space-shuttle-model.tar（预先封装的模型）上传到 HDFS：

(2) 创建所需的服务实例（如果它们尚不存在）。应用程序将使用 Spring Cloud 连接器连接到这些服务实例。

注意：如果您对所需的服务实例使用推荐的名称，当将它们推送到 Cloud Foundry 时它们会自动与应用程序绑定。否则，将需要在“manifest.yml”文件中调整服务实例名称或者在将应用程序推送到 Cloud Foundry 之后手动从“manifest.yml”中删除并进行绑定。

(3) 创建 Java 程序包：mvn package

(4)（可选）如果您未使用推荐的名称创建服务实例，请编辑自动生成的“manifest.yml”文件，在服务部分中调整服务实例的名称以便与您创建的服务实例的名称相匹配。您也可以删除服务部分，随后手动绑定它们。您还可能希望更改应用程序的主机或名称，同样可以在这个文件中修改。

(5) 使用 Cloud Foundry (CF) 命令“cf push”将该应用程序推送到平台。

(6) （可选）如果您从“manifest.yml”中删除了服务部分，则该应用程序尚不会启动。首

先，将所需的服务实例绑定到应用程序（“cf bind-service”），然后再现该应用程序（“cf restage”）。

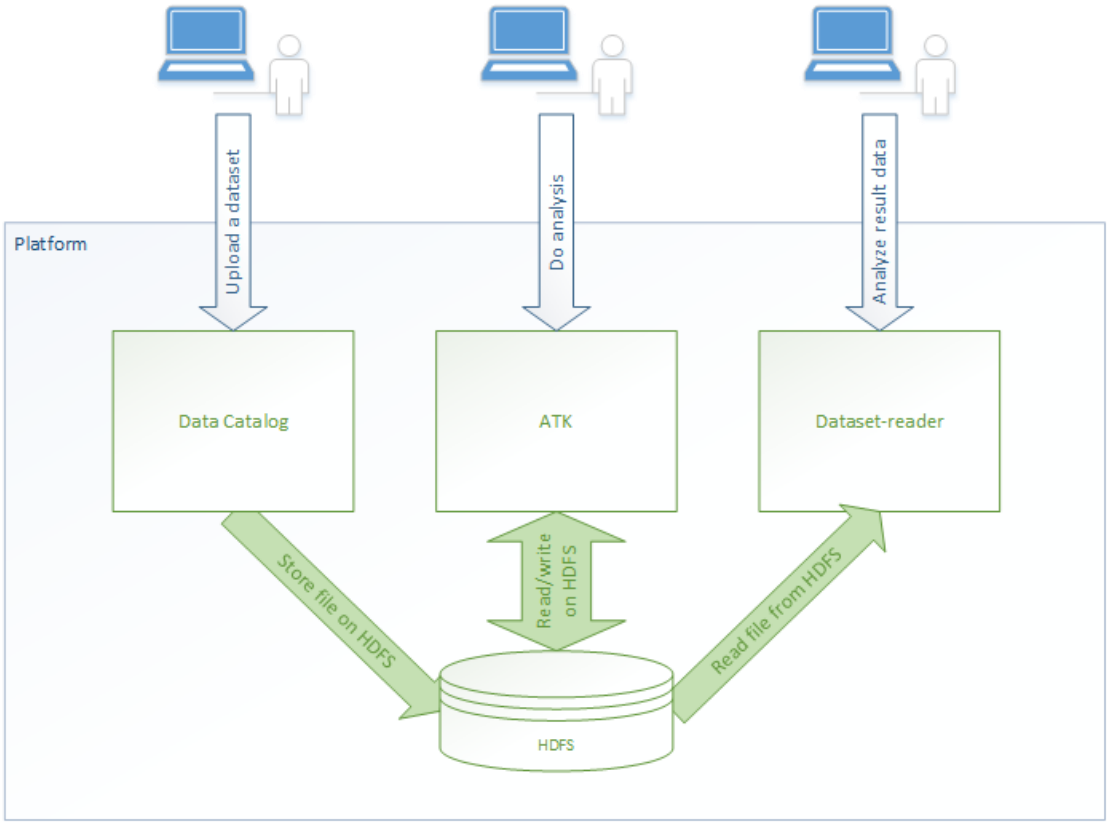
(7) 该应用程序现在已启动并正在运行。您应该会在显示的异常数据前面看到航天飞机图像。

1.3 将数据发送到 Kafka

若要通过网关将数据发送到 Kafka，您可以将 space_shuttle_client 从 client 目录推送到具有现有网关实例的空间或使用 Python 文件“space_shuttle_client.py”将 Gateway url 作为参数在本地传递。

2. 数据集读取器示例

该项目包含一个示例应用程序，该应用程序从 HDFS 中读取数据集并以图形形式将数据集呈现给用户。下图显示了此示例的数据流。



图片 61 数据读取案例图示

2.1 实现概要

(1) 数据集通过 I9000 数据目录上传到平台并存储在 HDFS 上。

(2) 数据科学家使用分析工具包对其进行分析。结果文件也存储在 HDFS 上。

(3) 应用程序开发人员将数据集读取器应用程序上传到平台并将其与数据科学家产生的文件绑定在一起。

(4) 数据集读取器应用程序以图表集的形式显示数据集。

2.2 准备数据

了解平台的数据分析功能的最佳方法是运行示例程序第一部分 对数据进行分析。但是，

如果您对使用应用程序可视化数据更感兴趣，您可以遵循下面的步骤以便使用预先构建的数据集。

2.2.1 使用预先构建的数据集

(1) 复制这个示例库。

(2) 在 I9000 控制台上，导航到“数据目录”>“提交转移”。

(3) 选择本地路径，然后导航到本地文件并选择该文件。

(4) 选择要上传的文件（可以在此处找到示例数据集：data/nf-data-application.cdv）。或者您也可以选择“链接”并将该链接粘贴到网页上的原始文件（即 nf-data-application.csv）。

(5) 在“标题”字段中输入标题。

(6) 单击“上传”按钮。

(7) 转移完成之后，您的新数据集将在“数据目录”>“数据集”中可见。

(8) 若要获取 HDFS 上的文件链接，请在“数据目录”>“数据集”中单击您的数据集的名称并复制 targetUri 属性的值。

2.3 使用应用程序可视化数据

若要了解平台的数据可视化功能，请运行示例程序的第二部分：可视化数据。

注意：如果您使用预先构建的数据集（而不是构建您自己的数据集），您仍然可以使用“可视化数据”。但请记住，您需要提供数据集的保存路径，即上面提到的 targetUri（更新环

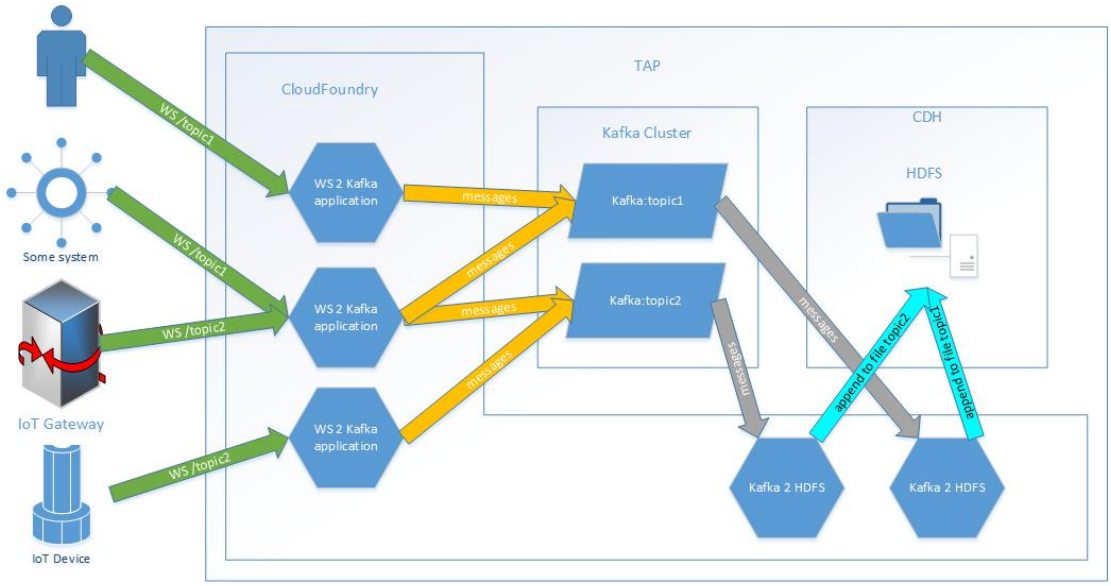
境并设置数据集链接）。

若不使用预先构建的数据集，您应该先训练数据集：nf-hour.csv，从而生成可视化需要的数据。

在 Jupyter Notebook 中运行 Netflow_Demo.ipynb 文件，以完成数据集的训练。

3. 通过 Kafka 队列获取数据

该存储库包含一个展示如何连接 I9000 服务的示例应用程序。下图显示了使用 Apache Kafka 获取数据的示例流程。



图片 62 kafka 获取数据

3.1 实现概要

(1) 外部设备/系统通过 I9000 中的 WebSockets (WS) 应用程序推送数据。让我们调用这个应用程序 ws2kafka。

(2) ws2kafka 将收到的数据推送到 Kafka 中。Kafka 主题是基于 WS URL 选择的，因此

调用应如下所示：wss://ws2kafka.some_domain.com/topic1。若要让该应用程序工作，必须将 Kafka 配置为自动创建主题。ws2kafka 可水平扩展，因此您可以有多个实例。

(3) kafka2hdfs 是一个初始具有跟踪预先定义的 Kafka 主题列表的应用程序。对于每个主题，它都确保存在相应的文件并且向该文件中追加新添加的数据。（由于向一个 HDFS 文件同时进行多次追加不安全，因此您应该避免在同一个主题上监听多个实例。）

有关如何部署每个应用程序的说明都包含在该应用程序的文件夹中 ,而且还有其他详细信息,例如 ,这里有一个小的实用工具脚本 create_service_instances.sh ,它创建服务实例。有关使用 I9000 控制台创建实例的说明，请访问此文件。

提供的 ws2kafka 应用程序非常简单，它不提供身份验证。如果您需要更高级的内容或者只是觉得有风险，您可以考虑使用 Gateway，但请注意，它使用的 Kafka 消息格式。

这个管道的一个非常方便的功能是，您可以将它的一部分替换成您自己的内容。

八、GearPump

Apache Gearpump

Apache Gearpump 是一个使用 Akka 的轻量级实时大数据流引擎。它也可以用作其他一般分布式应用程序的底层平台，如分布式 cron 作业或分布式日志收集。Apache Gearpump 的灵感来自于 Akka 框架的最新进展以及改进现有流式框架的愿望。Apache Gearpump 利用很多现有的框架，包括 MillWheel、Apache Storm、Spark Streaming、Apache Samza、Apache Tez 和 Hadoop YARN，同时在其整个架构中都利用了 Akka actors。

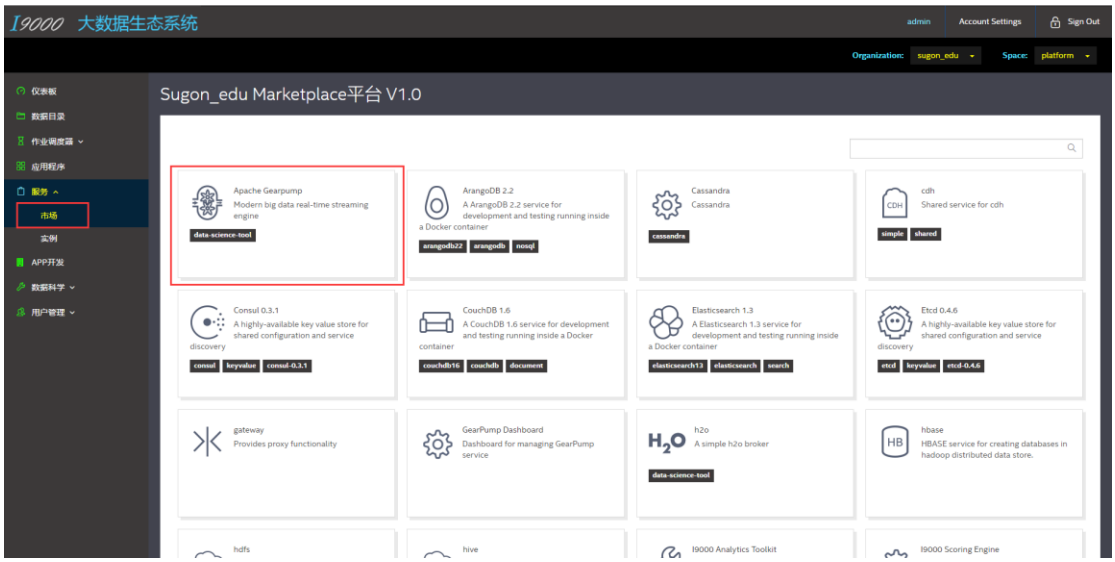
注意：详细内容，请参阅 http://gearpump.apache.org 上的文档。

Apache Gearpump 可以单独使用，也可以与大数据分析平台（I9000）完美结合。它可用于高速数据获取、流数据处理、在线分析等。目前，根据需要您可以通过 I9000“服务市场”选项卡创建一个新的 Apache Gearpump 集群。（Master 和 worker 部署到 YARN，Apache Gearpump 仪表板作为 Cloud Foundry 应用程序运行。）Apache Gearpump 实例可应用于给定组织的所有用户。

I9000 控制台在“数据科学”选项卡中还有 Apache Gearpump 部分。您可以在这里看到已有的服务实例。也可以通过 I9000 控制台直接部署 Apache Gearpump 应用程序。

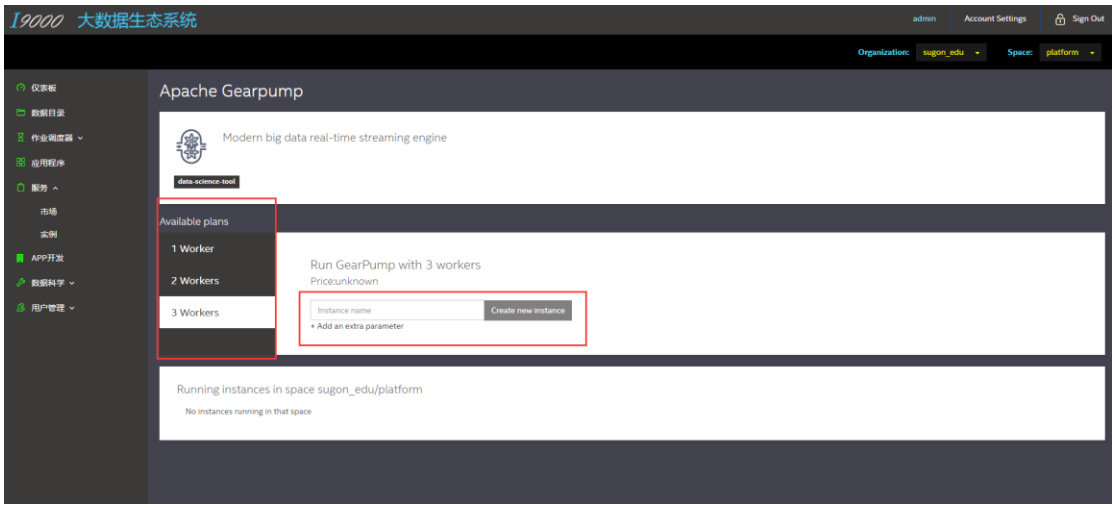
1. 创建 Apache Gearpump 实例

如果您想在 I9000 上创建新的 Apache Gearpump 实例，请转到“服务”>“市场”选项卡并选择“Apache Gearpump 服务”：



图片 63 市场

选择您要使用多少 worker（默认值 = 1）并指定实例名称：



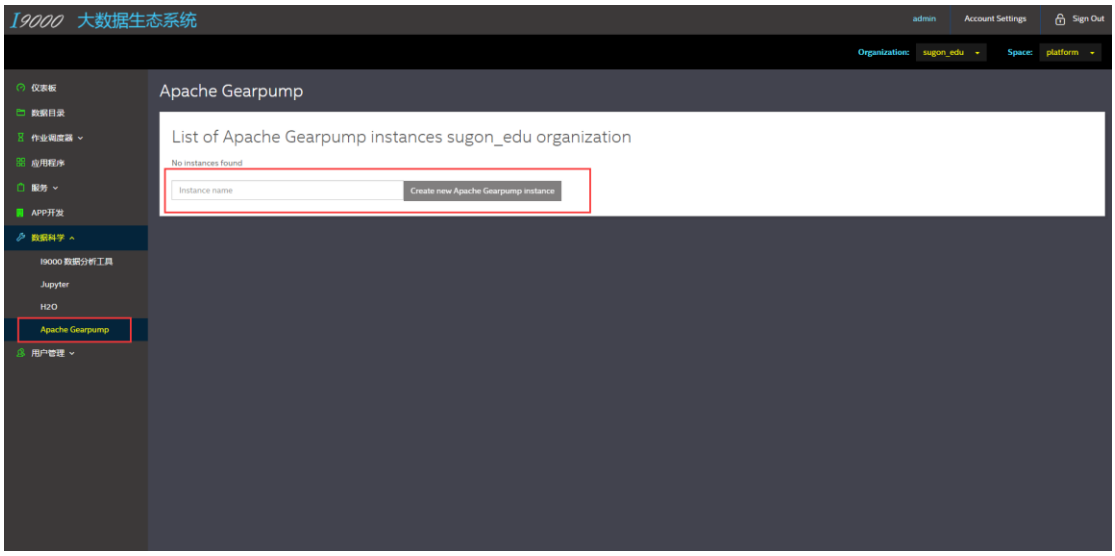
图片 64 创建梳理

单击“创建新实例”。

请耐心等待，创建新的 Gearpump 实例可能需要几分钟时间。几分钟之后刷新页面并检查实例列表中是否有新的 Gearpump 实例。

提示：

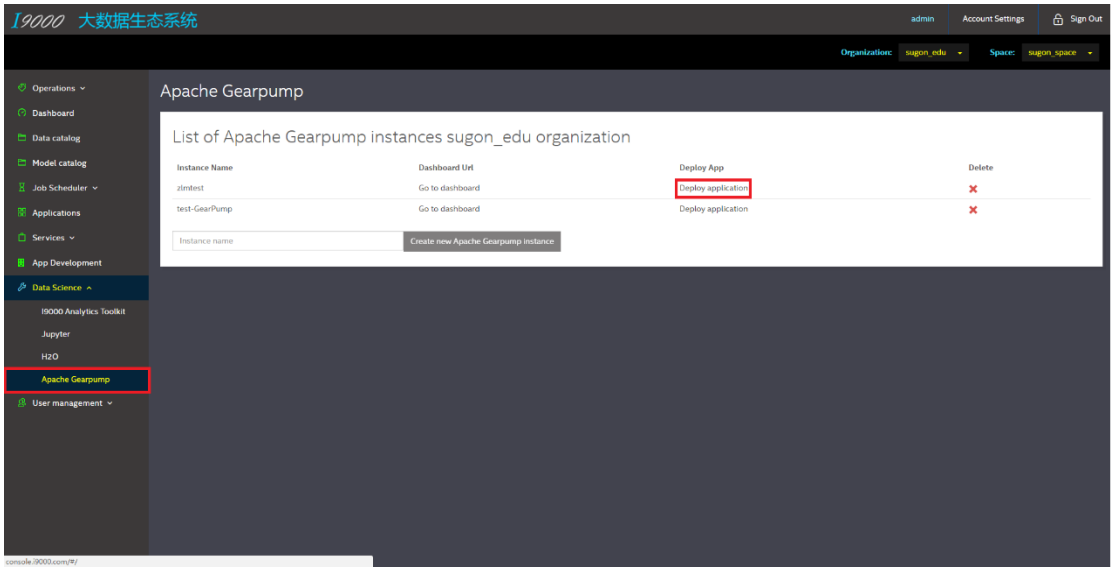
您也可以从 Apache Gearpump 实例列表中直接创建新实例。转到“数据科学”>“Apache Gearpump”，输入实例名称并单击“创建新的 Apache Gearpump 实例”按钮。这个新实例将有 1 个 worker（默认值）。



图片 65 在数据科学中创建

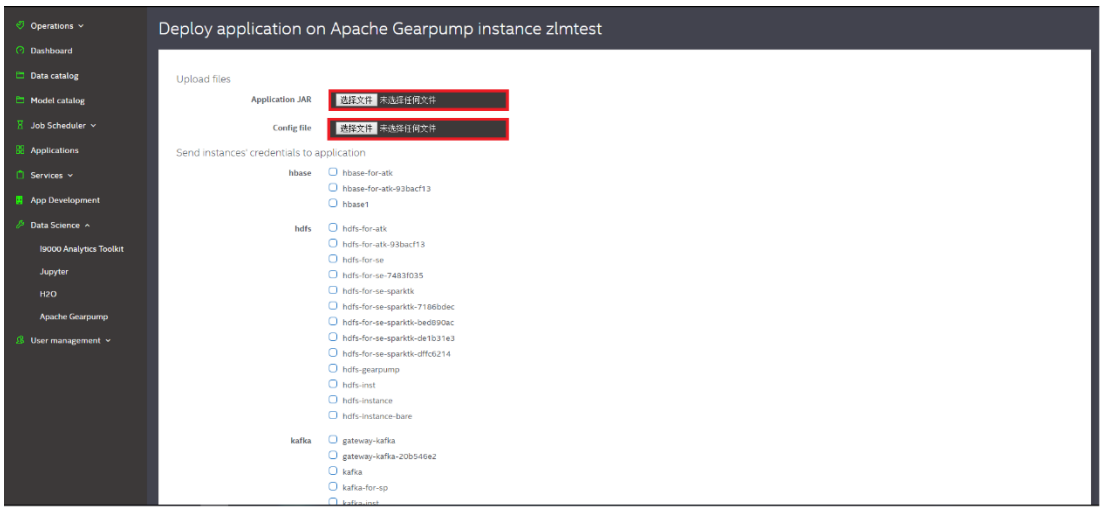
1. 在 Apache Gearpump 上部署应用程序

若要在 Apache Gearpump 上部署应用程序，您必须首先创建该服务的实例。创建之后，导航到“数据科学”>“Apache Gearpump”选项卡。然后单击您的实例旁边的“部署应用程序”。



图片 66 部署应用程序

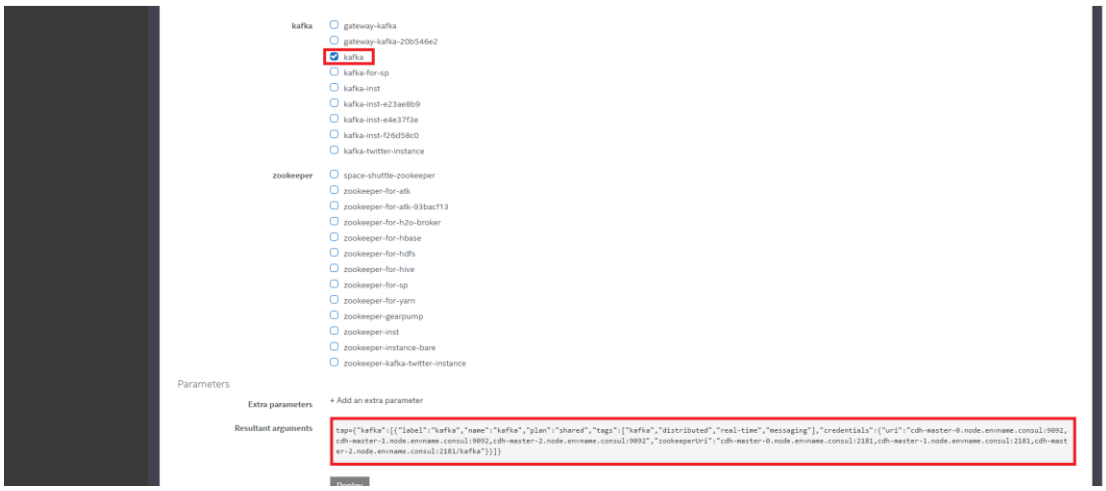
现在，选择您的应用程序 jar 文件。也可以添加配置文件（可选）。



图片 67 添加配置文件

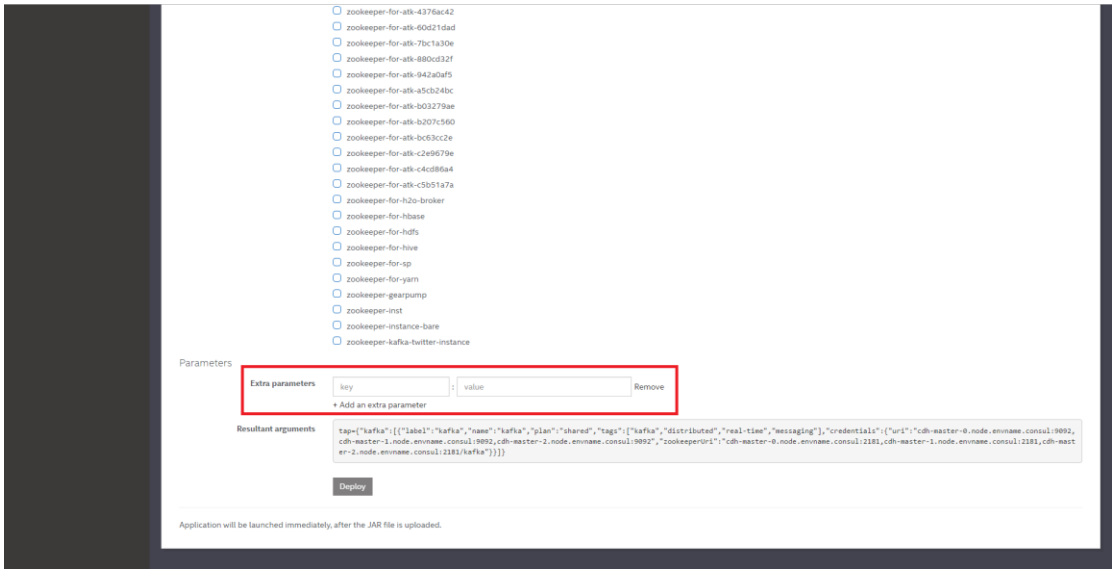
若要绑定现有的 I9000 服务到您的应用程序，请选中服务实例名称旁边的复选框（可选）。

将发送到您的应用程序的服务数据到一个“结果参数”字段。



图片 68 结果字段

您也可以使用键值对为您的应用程序添加一些额外的参数，例如 Kafka 主题名称。默认情况下显示第一对的字段。对于您要提供的每个键值对，单击“+ 添加额外参数”，然后输入该键值对的信息。

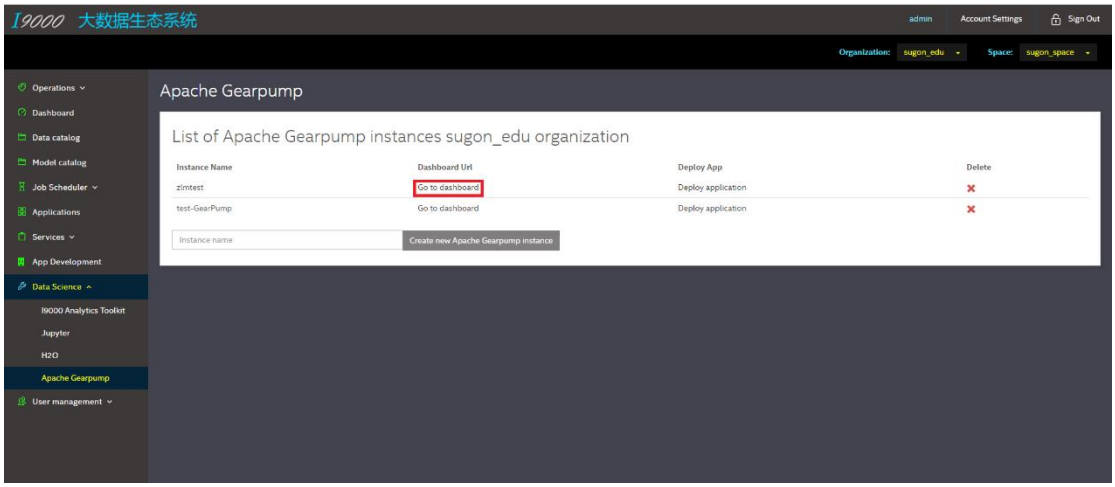


图片 69 添加参数

检查结果参数，如果您觉得它们都正确，请单击“部署”按钮以完成部署。

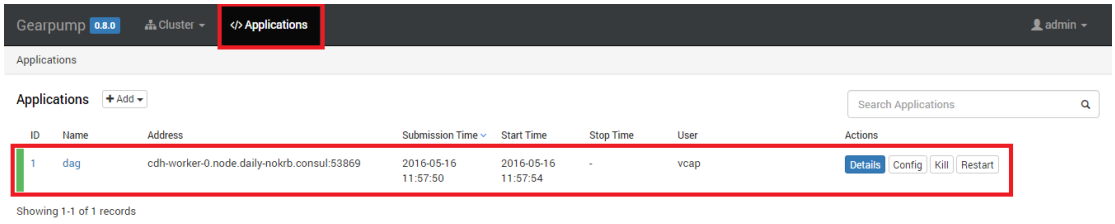
一般来说，部署时间取决于您的环境和 jar 大小。 对于大多数示例，通常需要不到一分钟。

成功完成部署之后，您将返回到 Apache Gearpump 屏幕。选择“转到仪表板”。



图片 70 仪表板

当显示仪表板时，导航到“应用程序”选项卡。您将看到您的应用程序位于列表上。



图片 71 应用程序列表

九、附录

ArangoDB 概述

ArangoDB 是一个开源 NoSQL 数据库，官网：https://www.ArangoDB.org/。

ArangoDB 支持灵活的数据模型，比如文档 Document、图 Graph 以及键值对 Key-Value 存储。ArangoDB 同时也是一个高性能的数据库，它使用类 SQL 查询或 JavaScript 扩展来构建高性能应用。

ArangoDB 值得称赞的一点，可以在树莓派上运行 ArangoDB 1.4 版。

ArangoDB 的特性：

1）多模型数据库

可以灵活的使用键值对、文档、图及其组合构建你的数据模型。

2）查询便利

ArangoDB 有类 SQL 的 AQL 查询语言，还可以通过 REST 方式进行查询。

3）可通过 JavaScript 进行扩展

无语言范围的限制，可以从前端到后端都使用同一种语言。

4）高性能

ArangoDB 速度极快

5）Foxx - 构建自己的 API

用 JavaScript 和 ArangoDB 构建应用，Foxx 运行在 DB 内部，可快速访问数据。

6）空间利用率高

跟其它文档型数据库相比，ArangoDB 占用的存储空间更少，因为 ArangoDB 是模式自由的元数据模式。

7）简单易用

ArangoDB 可以在几秒内启动运行，同时可使用图形界面来管理你的 ArangoDB。

8）多 OS 支持

ArangoDB 支持 Windows、Linux 和 OSX 等操作系统，还支持树莓派。

9）开源且免费

ArangoDB 开源免费，它采用了 Apache 2 许可证协议。

10）复制

ArangoDB 支持主从集群

ArangoDB 程序介绍

1、arangod

它是 ArangoDB 数据库的守护程序，运行后就是 ArangoDB 数据库服务器的守护进程。

2、arangosh

ArangoDB 的 Shell 环境。

3、arangoimp

ArangoDB 数据库导入工具

4、arangodump

ArangoDB 数据库的备份工具

5、arangorestore

ArangoDB 数据库的恢复工具

6、foxx-manager

一个 Shell 脚本，管理 Foxx 应用程序

7、arango-dfdb

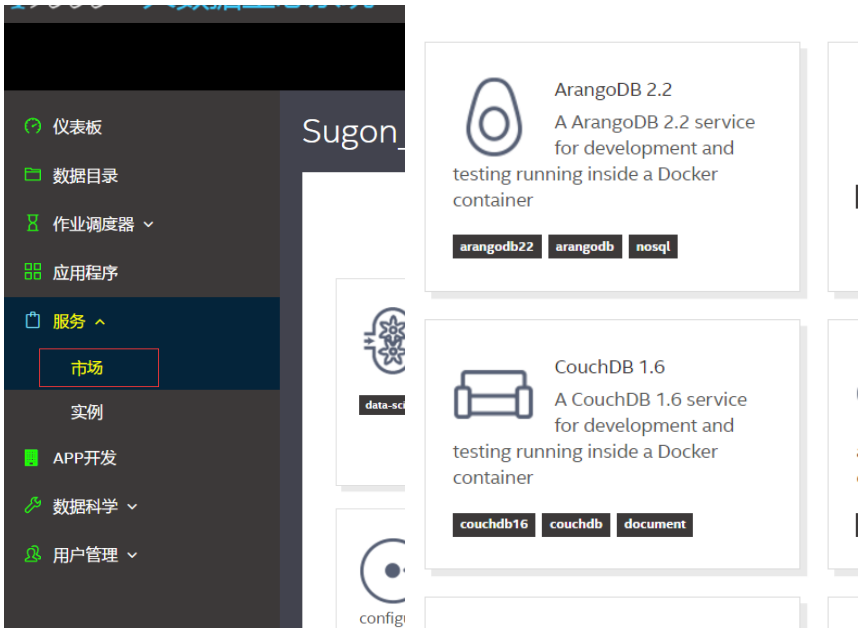
ArangoDB 的数据文件调试器

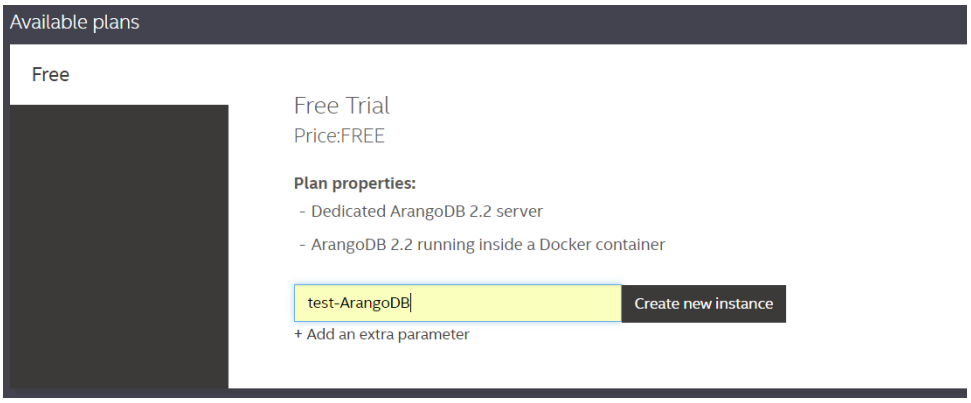
8、arangob

ArangoDB 的测试和评分工具，主要用于 ArangoDB 的开发和测试。

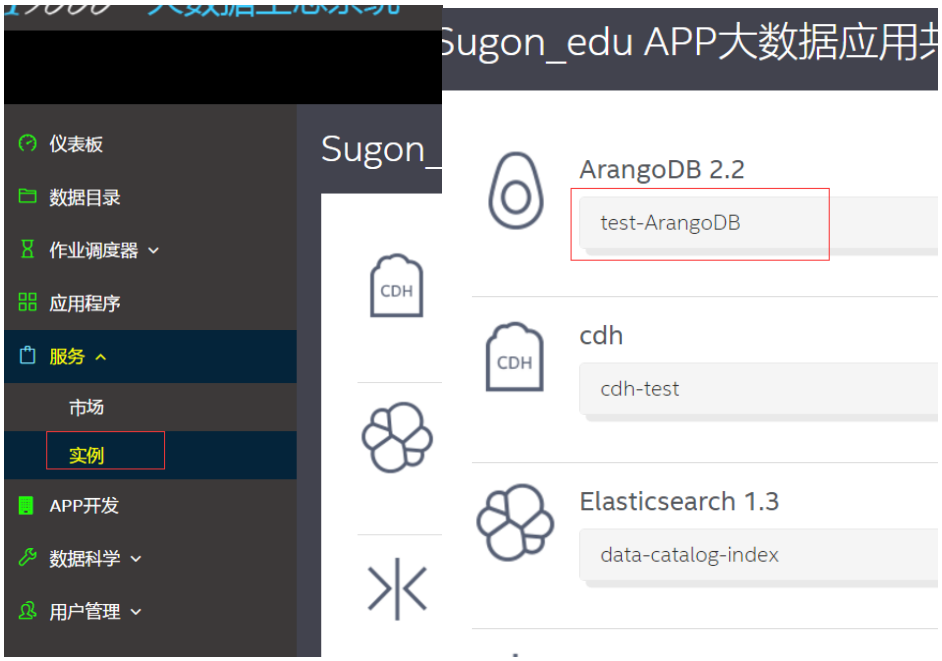
使用方法：

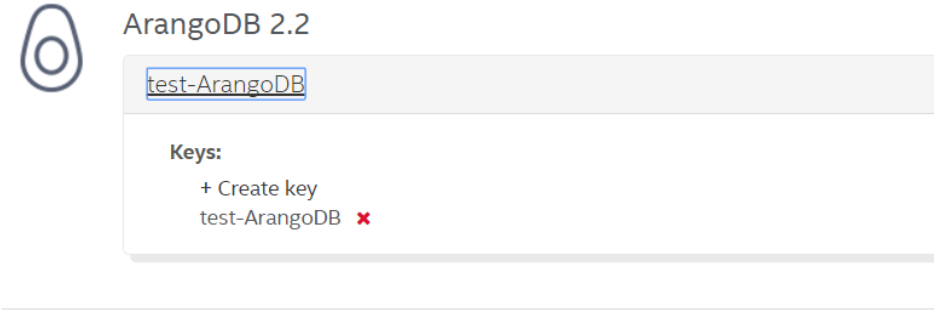
1. 在 i9000 平台上选择 服务>市场，找到 ArangoDB，点击 ArangoDB，创建一个新的实例。



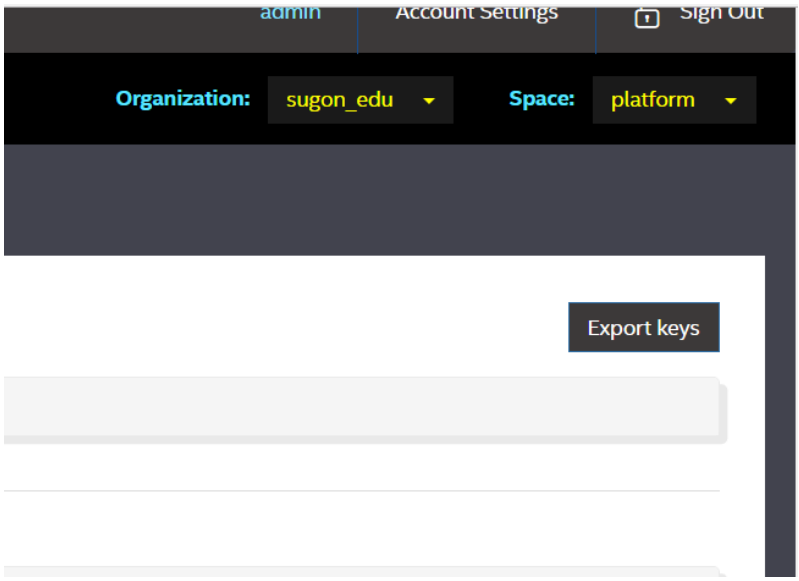


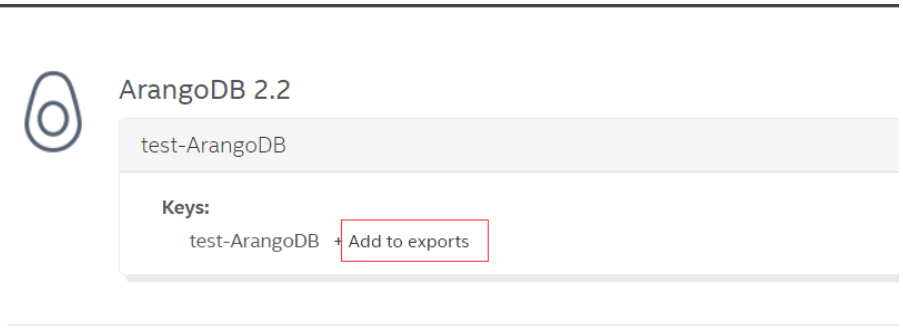
2. 服务>实例 里面找到 ArangoDB，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 ArangoDB 的连接信息。

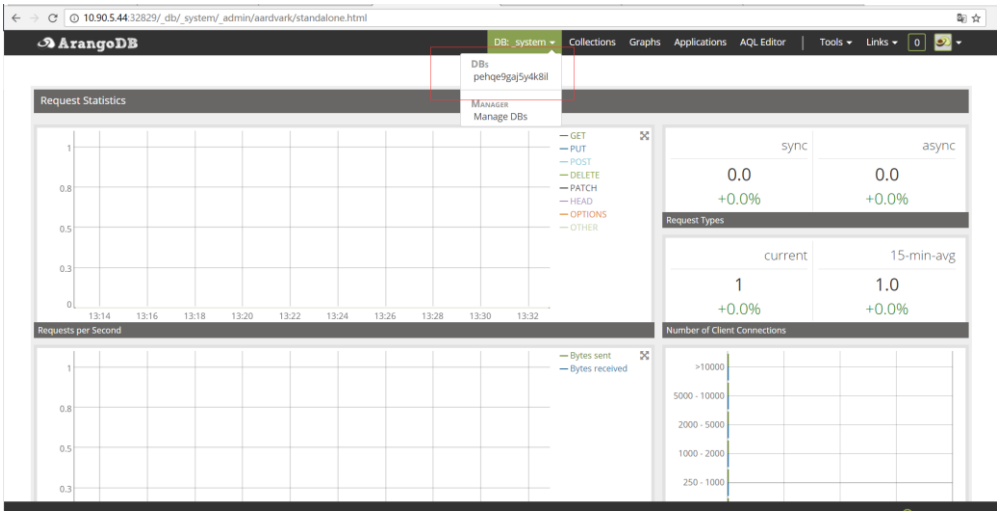




exported keys:


```
{
  "arangodb22": [
    {
      "label": "arangodb22",
      "name": "test-ArangoDB",
      "plan": "free",
      "tags": [
        "arangodb22",
        "arangodb",
        "nosql"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "8529/tcp": "32829"
        },
        "port": "32829",
        "username": "otyfsfluaicoy2yt",
        "password": "bcrugijqjzuhw43q",
        "dbname": "pehqe9gaj5y4k8il",
        "uri": "arangodb://otyfsfluaicoy2yt:bcrugijqjzuhw43q@10.0.4.4:32829/pehqe9gaj5y4k8il"
      }
    }
  ]
}
```


4. 在浏览器里输入新建实例的 ip 地址和端口号进行访问 ,可以看到界面化的 ArangoDB。


10.90.5.44:32829/_db/_system/_admin/aardvark/standalone.html
ArangoDB
DB: _system
Collections
Graphs
Applications
AQL Editor
Tools
Links
0
DBs
pehqe9gaj5y4k8il
Manager
Manage DBs
Request Statistics
GET
PUT
POST
DELETE
PATCH
HEAD
OPTIONS
OTHER
sync
async
0.0
+0.0%
0.0
+0.0%
Request Types
current
15-min-avg
1
+0.0%
1.0
+0.0%
Requests per Second
Number of Client Connections
Bytes sent
Bytes received
>10000
5000 - 10000
2000 - 5000
1000 - 2000
250 - 1000

Cassandra 概述

Apache Cassandra 是一套开源分布式 Key-Value 存储系统。它最初由 Facebook 开发，用于储存特别大的数据。 Cassandra 不是一个数据库，它是一个混合型的非关系的数据库，类似于 Google 的 BigTable。

Cassandra 的数据存储结构

Cassandra 的数据模型是基于列族（Column Family）的四维或五维模型。它借鉴了 Amazon 的 Dynamo 和 Google's BigTable 的数据结构和功能特点，采用 Memtable 和 SSTable 的方式进行存储。在 Cassandra 写入数据之前，需要先记录日志（CommitLog），然后数据开始写入到 Column Family 对应的 Memtable 中，Memtable 是一种按照 key 排序数据的内存结构，在满足一定条件时，再把 Memtable 的数据批量的刷新到磁盘上，存储为 SSTable。

Cassandra 的数据模型的基本概念：

1. Cluster：Cassandra 的节点实例，它可以包含多个 Keyspace
2. Keyspace ：用于存放 ColumnFamily 的容器，相当于关系数据库中的 Schema 或 database
3. ColumnFamily：用于存放 Column 的容器，类似关系数据库中的 table 的概念
4. SuperColumn ：它是一个特列殊的 Column，它的 Value 值可以包函多个 Column
5. Columns：Cassandra 的最基本单位。由 name，value，timestamp 组成

Keyspace:Keyspace1

ColumnFamily:Standard2

Key	Columns

studentA

Columns		
name	value	timestamp
"age"	"18"	1270694041669000
"height"	"172cm"	1270694041669000

classA

SuperColumns	
Key	Columns

studentX

Columns		
name	value	timestamp
"age"	"20"	1270694041669000
"height"	"182cm"	1270694041669000

数据模型实例分析

写入一个 studentA 的信息：

```
set Keyspace1.Standard2['studentA']['age'] = '18'
       \            \         \        \          \
        \            \         \_ key   \          \_ value
         \            \                  \_ column
          \_ keyspace  \_ column family
```

读取 studentA 列信息：

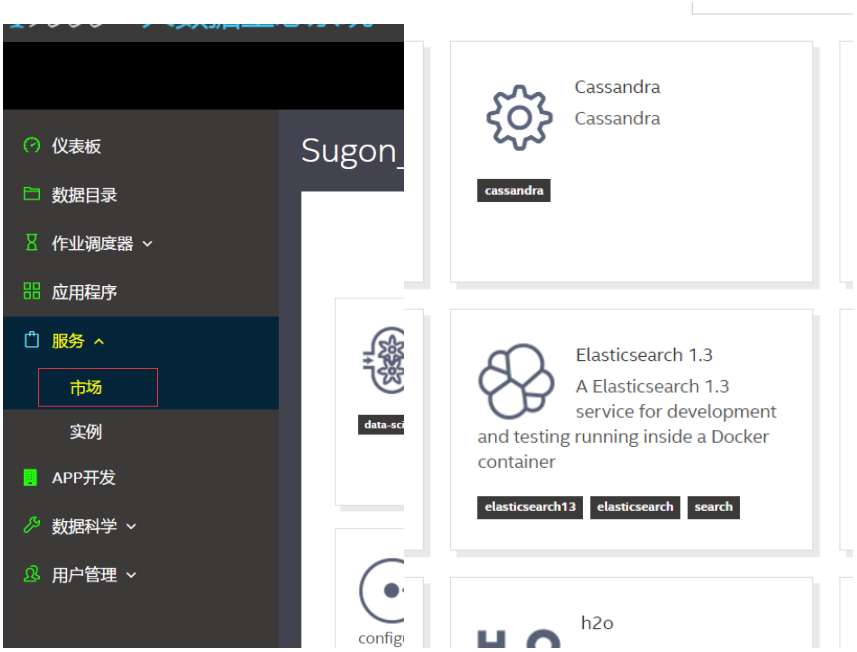
```
get Keyspace1.Standard2['studentA']
```

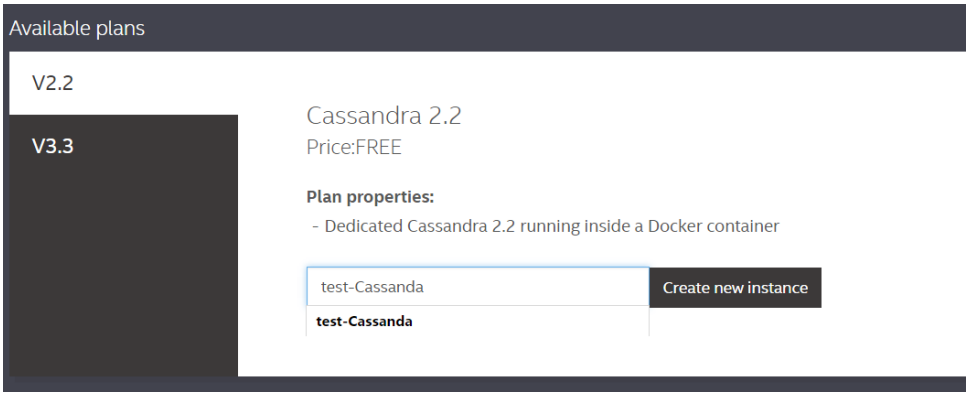
返回的结果：

```
(column=age, value=18, timestamp=1270694041669000)
```

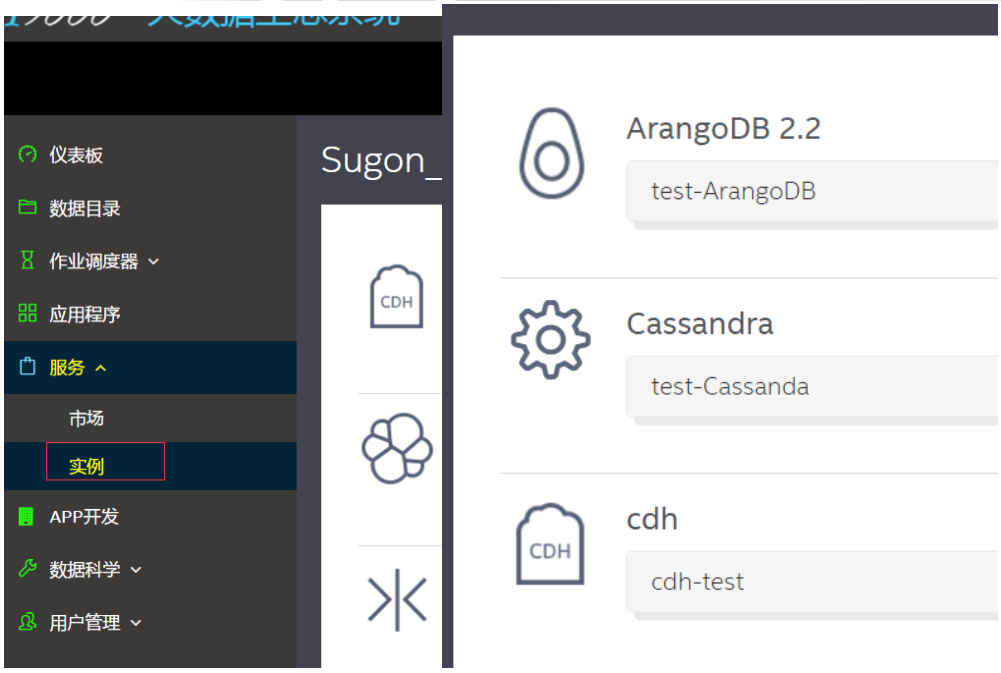
使用方法：

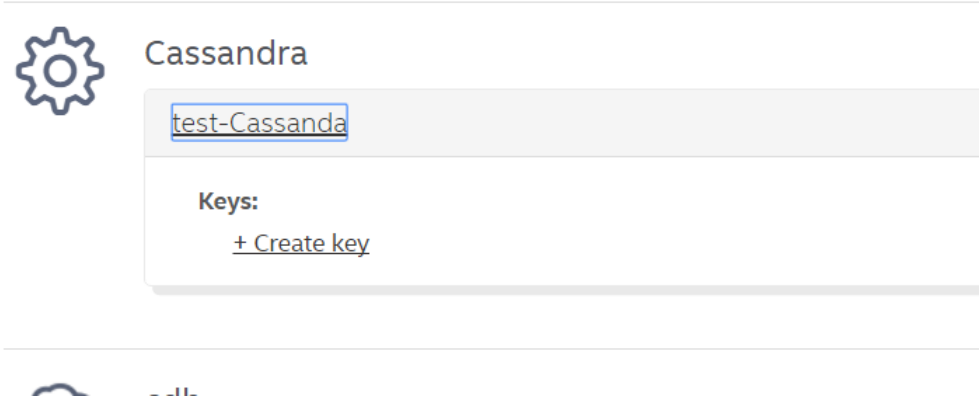
1. 在 i9000 平台上选择 服务>市场，找到 Cassandra，点击 Cassandra，创建一个新的实例。



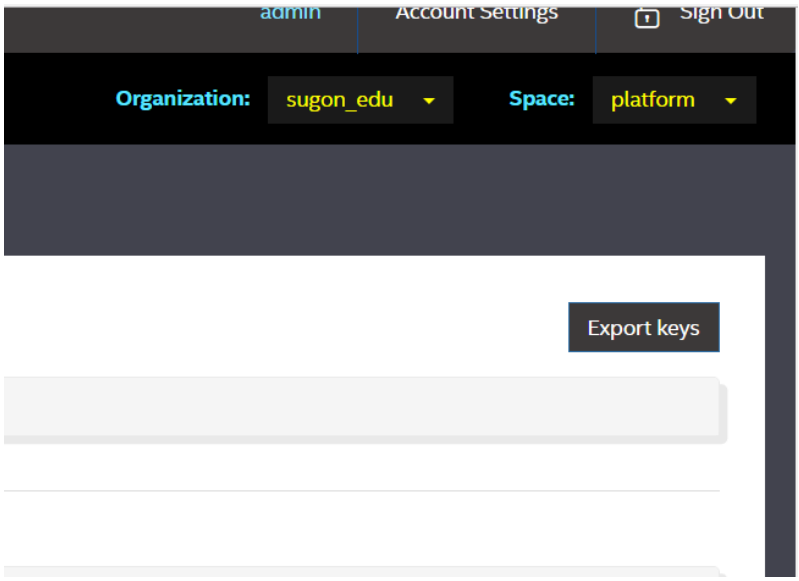


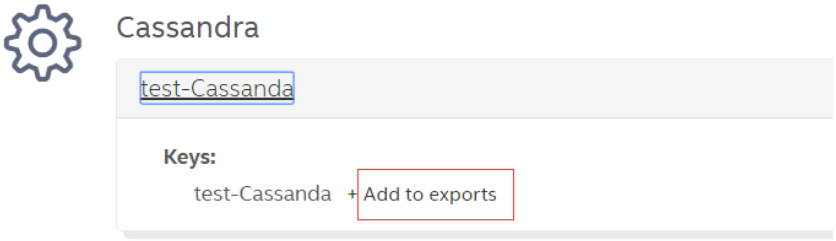
2. 在 服务>实例 里面找到 Cassandra，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Cassandra 的连接信息。





exported Keys:


```
{
  "cassandra": [
    {
      "label": "cassandra",
      "name": "test-Cassanda",
      "plan": "v2.2",
      "tags": [
        "cassandra"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "7000/tcp": "32830",
          "7001/tcp": "32831",
          "7199/tcp": "32832",
          "9042/tcp": "32833",
          "9160/tcp": "32834"
        }
      }
    }
  ]
}
```

4. 使用 hostname 的 ip 和 9160 映射的端口可以在代码中连接和使用新建的 Cassandra 数据库。

CDH 概述

Cloudera 的 CDH 和 Apache 的 Hadoop 的区别

目前而言，不收费的 Hadoop 版本主要有三个（均是国外厂商），分别是： Apache（最原始的版本，所有发行版均基于这个版本进行改进）、Cloudera 版本（Cloudera’s Distribution Including Apache Hadoop ,简称 CDH)、Hortonworks 版本(Hortonworks Data Platform ,简称“HDP”),对于国内而言 ,绝大多数选择 CDH 版本 ,CDH 和 Apache 版本主要区别如下：

(1) CDH 对 Hadoop 版本的划分非常清晰，只有两个系列的版本，分别是 cdh3 和 cdh4，分别对应第一代 Hadoop（Hadoop 1.0）和第二代 Hadoop（Hadoop 2.0），相比而言，Apache 版本则混乱得多；比 Apache hadoop 在兼容性，安全性，稳定性上有增强。

（补充：当前已有 CDH5，对应 Hadoop2.2.0 开始）

(2) CDH3 版本是基于 Apache hadoop 0.20.2 改进的，并融入了最新的 patch，CDH4 版本是基于 Apache hadoop 2.X 改进的,CDH 总是并应用了最新 Bug 修复或者 Feature 的 Patch，并比 Apache hadoop 同功能版本提早发布，更新速度比 Apache 官方快。

(3) 安全 CDH 支持 Kerberos 安全认证，apache hadoop 则使用简陋的用户名匹配认证

(4) CDH 文档清晰，很多采用 Apache 版本的用户都会阅读 CDH 提供的文档，包括安装文档、升级文档等。

(5) CDH 支持 Yum/Apt 包，Tar 包，RPM 包，CM 安装，Cloudera Manager 三种方式安装,Apache hadoop 只支持 Tar 包安装。

注：CDH 使用推荐的 Yum/Apt 包安装时，有以下几个好处：

1、联网安装、升级，非常方便

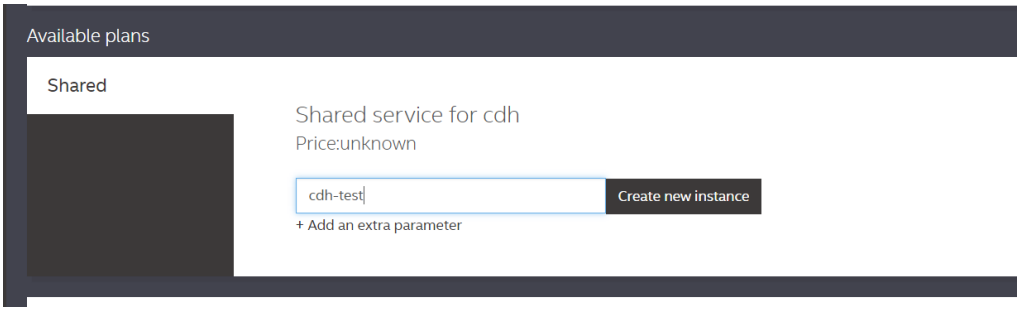
2、自动下载依赖软件包

3、Hadoop 生态系统包自动匹配，不需要你寻找与当前 Hadoop 匹配的 Hbase，Flume，Hive 等软件，Yum/Apt 会根据当前安装 Hadoop 版本自动寻找匹配版本的软件包，并保证兼容性。

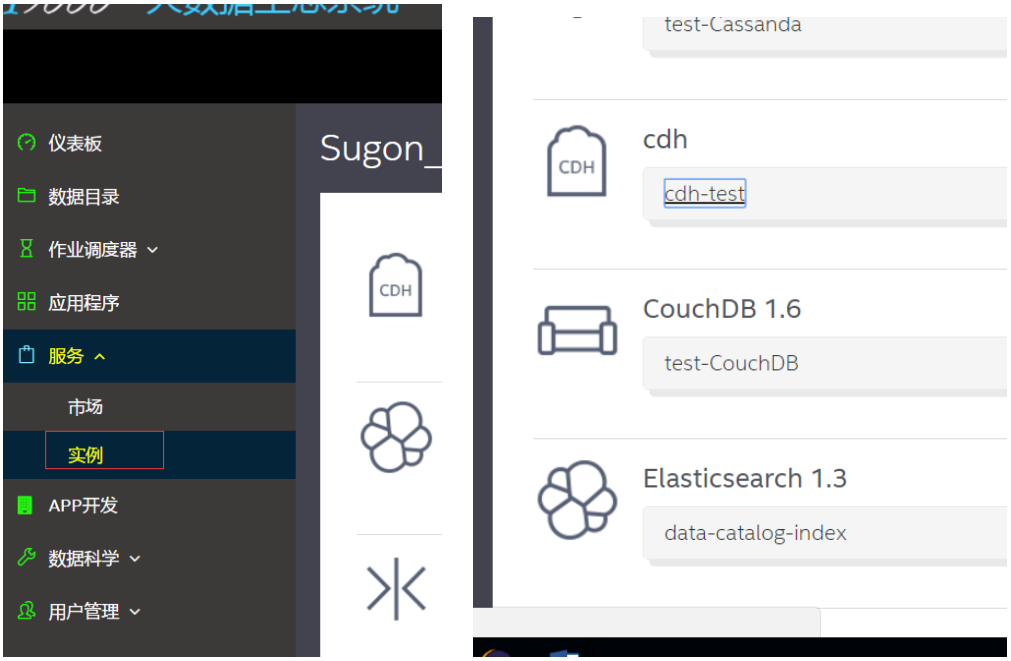
4、自动创建相关目录并软链到合适的地方（如 conf 和 logs 等目录）；自动创建 hdfs, mapred 用户，hdfs 用户是 HDFS 的最高权限用户，mapred 用户则负责 mapreduce 执行过程中相关目录的权限。

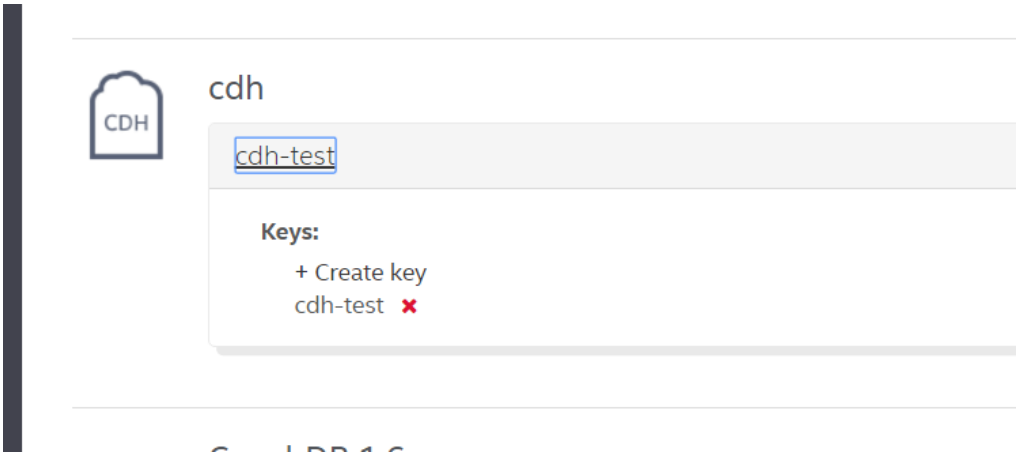
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 cdh，点击 cdh，创建一个新的实例。

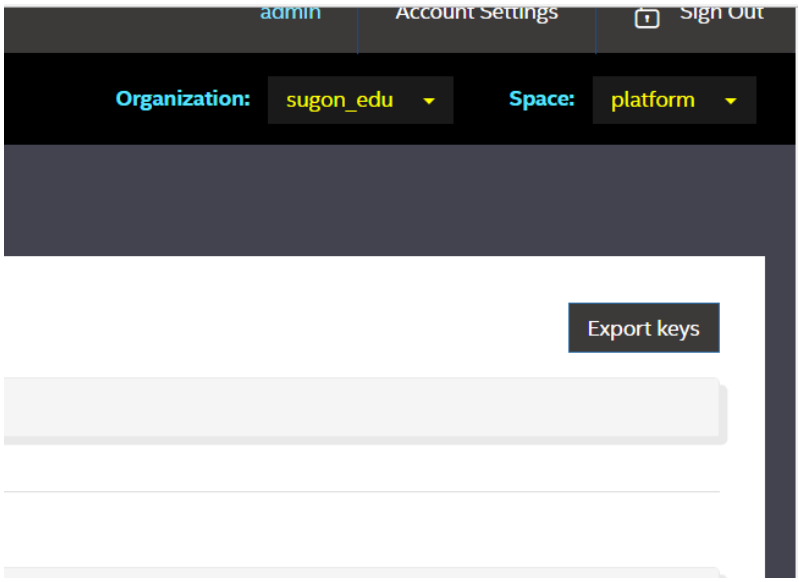


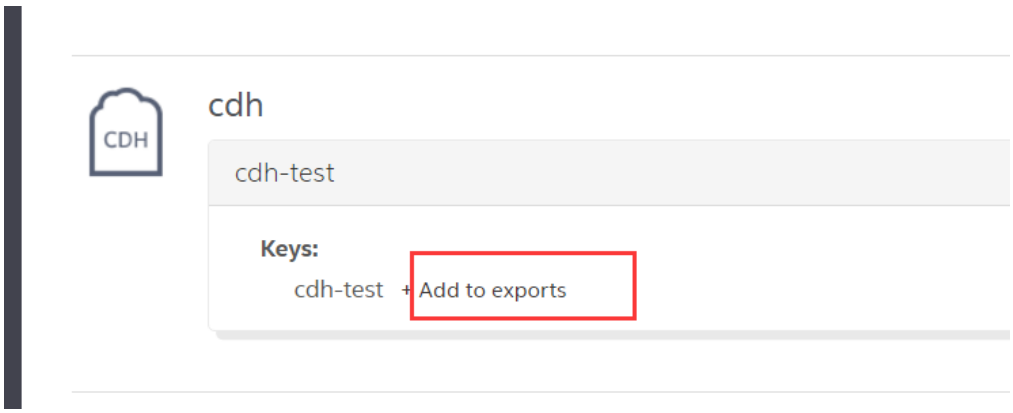
2. 在 服务>实例 里面找到 cdh，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 cdh 的配置信息。

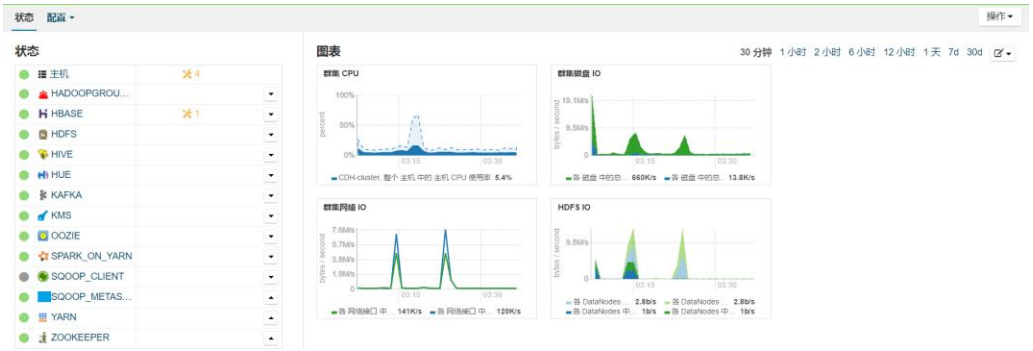




exported Keys:


```
{
  "cdh": [
    {
      "label": "cdh",
      "name": "cdh-test",
      "plan": "shared",
      "tags": [
        "simple",
        "shared"
      ],
      "credentials": {
        "hbase_config": "http://cdh-master-2.node.envname.consul:7180/api/v9/clusters/CDH-cluster/services/HBASE/clientConfig",
        "hdfs_config": "http://cdh-master-2.node.envname.consul:7180/api/v9/clusters/CDH-cluster/services/HDFS/clientConfig",
        "hdfs_root": "hdfs://nameservice1/user/atkuser",
        "resource_manager": "http://cdh-master-0.node.envname.consul:8088",
        "spark_max_memory": "2163212288",
        "yarn_config": "http://cdh-master-2.node.envname.consul:7180/api/v9/clusters/CDH-cluster/services/YARN/clientConfig",
        "zk_host": "cdh-master-0.node.envname.consul",
        "zk_port": "2181"
      }
    }
  ]
}
```


4. 这些配置信息提供了 Hbase、hdfs、yarn 等的配置文件获取路径，使用这些路径即可搭建 cdh。



Consul 0.3.1 概述

Consul 是一个支持多数据中心分布式高可用的服务发现和配置共享的服务软件,由 HashiCorp 公司用 Go 语言开发, 基于 Mozilla Public License 2.0 的协议进行开源. Consul 支持健康检查,并允许 HTTP 和 DNS 协议调用 API 存储键值对。

命令行超级好用的虚拟机管理软件 vgrant 也是 HashiCorp 公司开发的产品。

一致性协议采用 Raft 算法,用来保证服务的高可用. 使用 GOSSIP 协议管理成员和广播消息, 并且支持 ACL 访问控制。

Consul 的使用场景：

docker 实例的注册与配置共享

coreos 实例的注册与配置共享

vitess 集群

SaaS 应用的配置共享

与 confd 服务集成，动态生成 nginx 和 haproxy 配置文件

Consul 的优势：

使用 Raft 算法来保证一致性，比复杂的 Paxos 算法更直接. 相比较而言, zookeeper 采用的是 Paxos, 而 etcd 使用的则是 Raft。

支持多数据中心，内外网的服务采用不同的端口进行监听。 多数据中心集群可以避免单数据中心的单点故障,而其部署则需要考虑网络延迟, 分片等情况等. zookeeper 和 etcd

均不提供多数据中心功能的支持。

支持健康检查. etcd 不提供此功能。

支持 http 和 dns 协议接口. zookeeper 的集成较为复杂, etcd 只支持 http 协议。

官方提供 web 管理界面, etcd 无此功能。

综合比较, Consul 作为服务注册和配置管理的新星, 比较值得关注和研究。

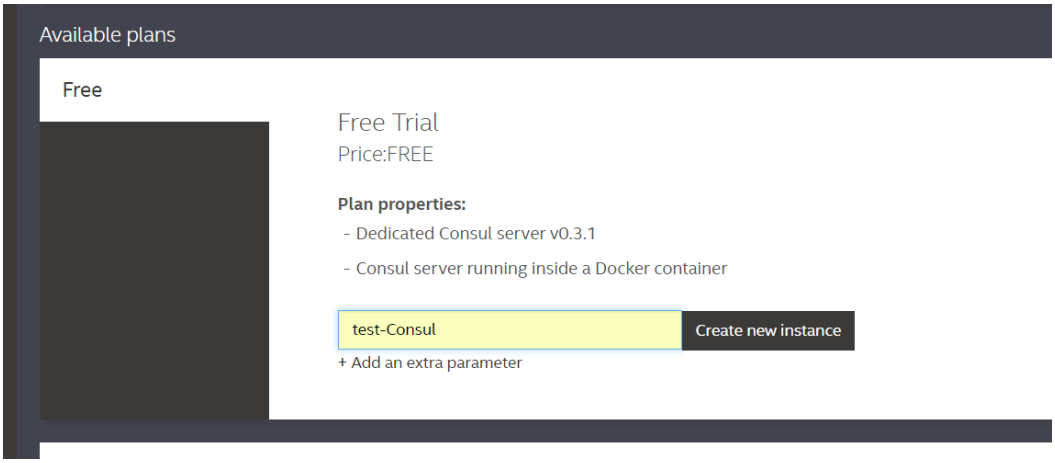
Consul 的角色

client: 客户端, 无状态, 将 HTTP 和 DNS 接口请求转发给局域网内的服务端集群.

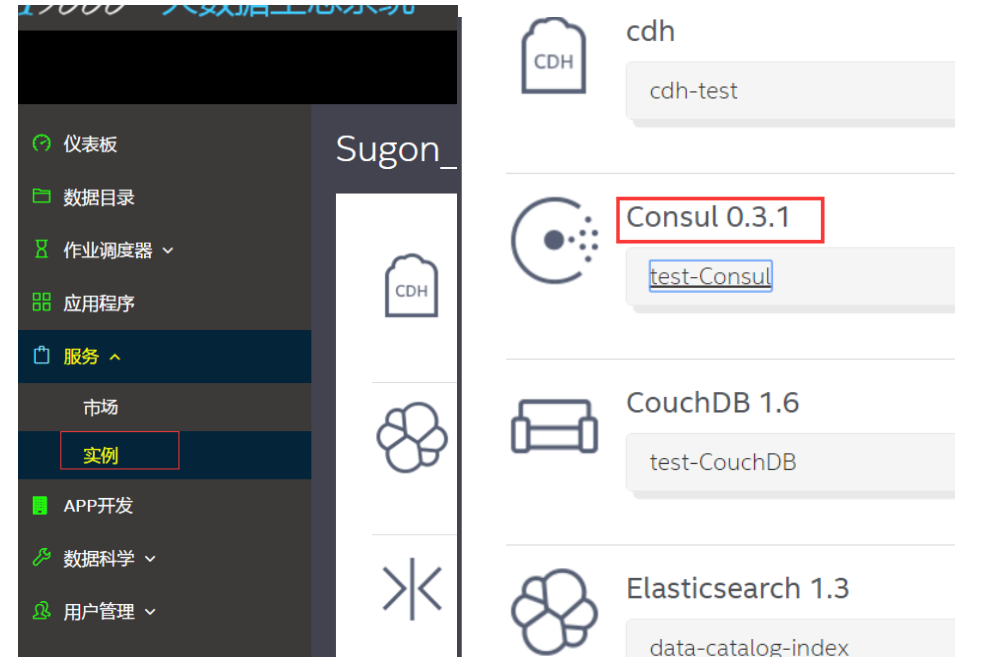
server: 服务端, 保存配置信息, 高可用集群, 在局域网内与本地客户端通讯, 通过广域网与其他数据中心通讯. 每个数据中心的 server 数量推荐为 3 个或是 5 个。

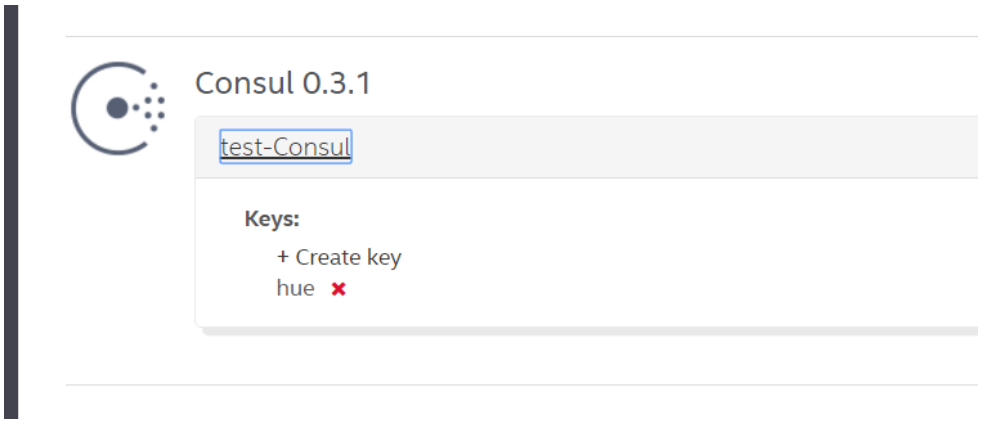
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 Consul 0.3.1，点击 Consul 0.3.1，创建一个新的实例。

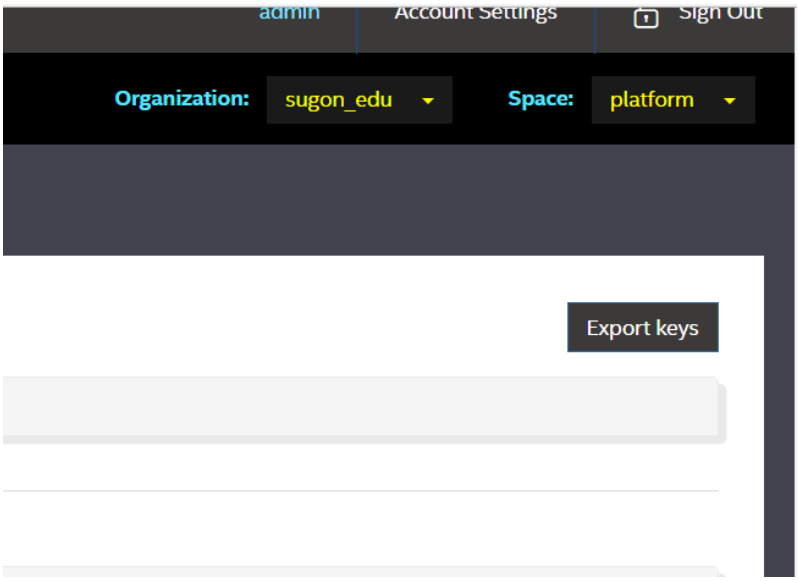


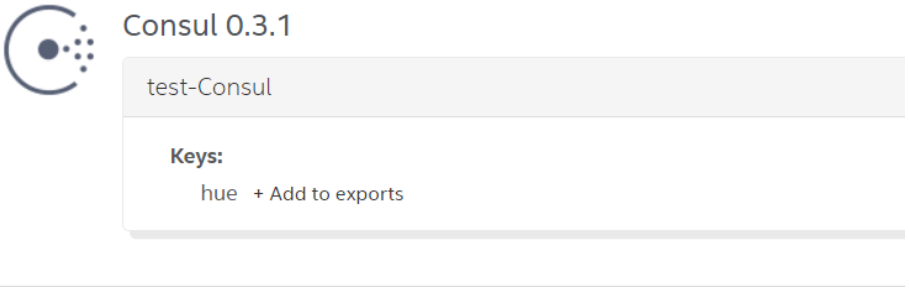
2. 在 服务>实例 里面找到 Consul 0.3.1，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Consul 0.3.1 的配置信息。

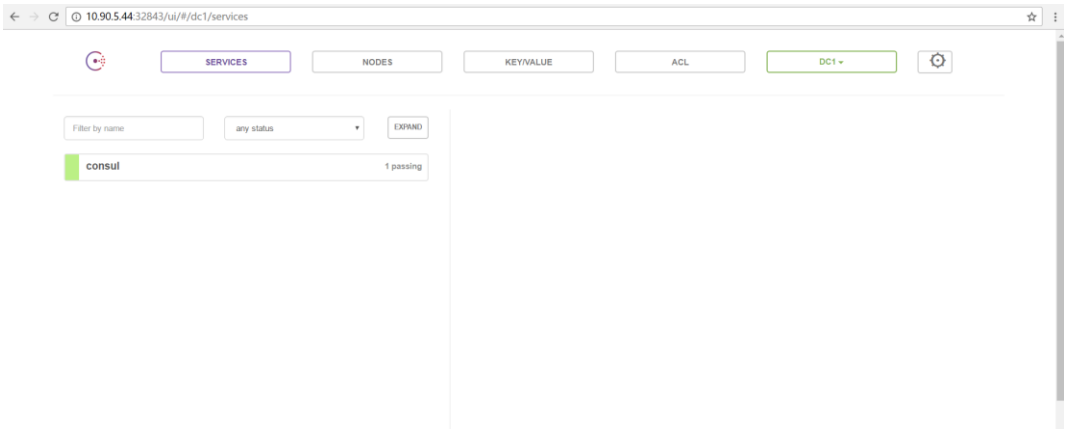




exported Keys:


```
{
  "consul": [
    {
      "label": "consul",
      "name": "test-Consul",
      "plan": "free",
      "tags": [
        "consul",
        "keyvalue",
        "consul-0.3.1"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "53/tcp": "32838",
          "53/udp": "32768",
          "8300/tcp": "32839",
          "8301/tcp": "32840",
          "8301/udp": "32769",
          "8302/tcp": "32841",
          "8302/udp": "32770",
          "8400/tcp": "32842",
          "8500/tcp": "32843"
        },
        "port": "32843"
      }
    }
  ]
}
```

4. 使用这些配置信息进行对 Consul 0.3.1 的配置，即可使用平台上新建的 Consul 0.3.1 实例。可以用浏览器访问 http://ipaddress:8500/ui/#/dc1/services 访问 consul 的 ui 控制台界面。端口作用分别是 8400 (RPC), 8500 (HTTP), 8300、8301、8302，这三个是用于 Consul 内部通信使用。



CouchDB 概述

CouchDB 是用 Erlang 开发的面向文档的数据库系统，最近刚刚发布了 1.0 版本(2010 年 7 月 14 日)。CouchDB 不是一个传统的关系数据库，而是面向文档的数据库，其数据存储方式有点类似 lucene 的 index 文件格式，CouchDB 最大的意义在于它是一个面向 web 应用的新一代存储系统，事实上，CouchDB 的口号就是：下一代的 Web 应用存储系统。

CouchDB 可以安装在大部分 POSIX 系统上，包括 Linux® 和 Mac OS X。尽管目前还不正式支持 Windows®，但现在已经着手编写 Windows 平台的非官方二进制安装程序。CouchDB 可以从源文件安装，也可以使用包管理器安装 (比如在 Mac OS X 上使用 MacPorts)。

CouchDB 是一个顶级 Apache Software Foundation 开源项目，根据 Apache 许

可 V2.0 发布。这个开源许可允许在其他软件中使用这些源代码，并根据需要进行修改，但前提是遵从版权需知和免责声明。与许多其他开源许可一样，这个许可允许用户根据需求使用、修改和分发该软件。不一定由同一个许可包含所有修改，因为我们仅维护一个 Apache 代码使用许可需知。

特点：

一、CouchDB 是分布式的数据库，他可以把存储系统分布到 n 台物理的节点上面，并且很好的协调和同步节点之间的数据读写一致性。这当然也得靠 Erlang 无与伦比的并发特性才能做到。对于基于 web 的大规模应用文档应用，分布式可以让它不必像传统的关系数据库那样分库拆表，在应用代码层进行大量的改动。

二、CouchDB 是面向文档的数据库，存储半结构化的数据，比较类似 lucene 的 index 结构，特别适合存储文档，因此很适合 CMS，电话本，地址本等应用，在这些应用场合，文档数据库要比关系数据库更加方便，性能更好。

三、CouchDB 支持 REST API，可以让用户使用 JavaScript 来操作 CouchDB 数据库，也可以用 JavaScript 编写查询语句，我们可以想像一下，用 AJAX 技术结合 CouchDB 开发出来的 CMS 系统会是多么的简单和方便。

其实 CouchDB 只是 Erlang 应用的冰山一角，在最近几年，基于 Erlang 的应用也得到的蓬勃的发展，特别是在基于 web 的大规模，分布式应用领域，几乎都是 Erlang 的优势项目。

工作原理：

CouchDB 构建在强大的 B-树储存引擎之上。这种引擎负责对 CouchDB 中的数据进

行排序，并提供一种能够在对数均摊时间内执行搜索、插入和删除操作的机制。CouchDB 将这个引擎用于所有内部数据、文档和视图。

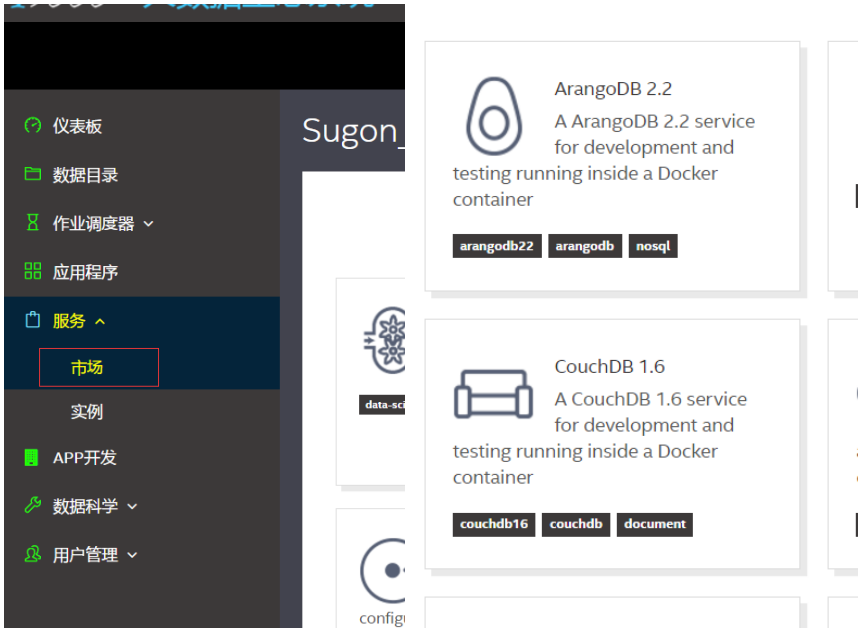
因为 CouchDB 数据库的结构独立于模式，所以它依赖于使用视图创建文档之间的任意关系，以及提供聚合和报告特性。使用 Map/Reduce 计算这些视图的结果，Map/Reduce 是一种使用分布式计算来处理和生成大型数据集的模型。Map/Reduce 模型由 Google 引入，可分为 Map 和 Reduce 两个步骤。在 Map 步骤中，由主节点接收文档并将问题划分为多个子问题。然后将这些子问题发布给工作节点，由它处理后再将结果返回给主节点。在 Reduce 步骤，主节点接收来自工作节点的结果并合并它们，以获得能够解决最初问题的总体结果和答案。

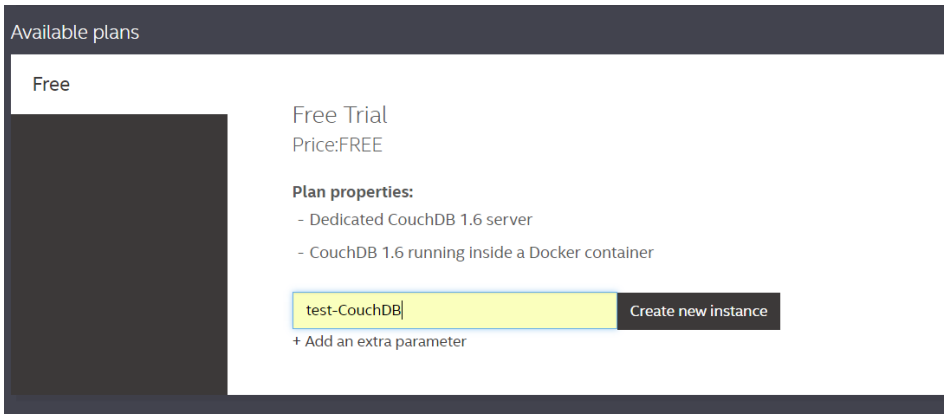
CouchDB 中的 Map/Reduce 特性生成键/值对，CouchDB 将它们插入到 B-树引擎中并根据它们的键进行排序。这就能通过键进行高效查找，并且提高 B-树中的操作的性能。此外，这还意味着可以在多个节点上对数据进行分区，而不需要单独查询每个节点。

传统的关系数据库管理系统有时使用锁来管理并发性，从而防止其他客户机访问某个客户机正在更新的数据。这就防止多个客户机同时更改相同的数据，但对于多个客户机同时使用一个系统的情况，数据库在确定哪个客户机应该接收锁并维护锁队列的次序时会遇到困难，这很常见。在 CouchDB 中没有锁机制，它使用的是多版本并发性控制（Multiversion concurrency controlMVCC）— 它向每个客户机提供数据库的最新版本的快照。这意味着在提交事务之前，其他用户不能看到更改。许多现代数据库开始从锁机制前移到 MVCC，包括 Oracle（V7 之后）和 Microsoft® SQL Server 2005 及更新版本。

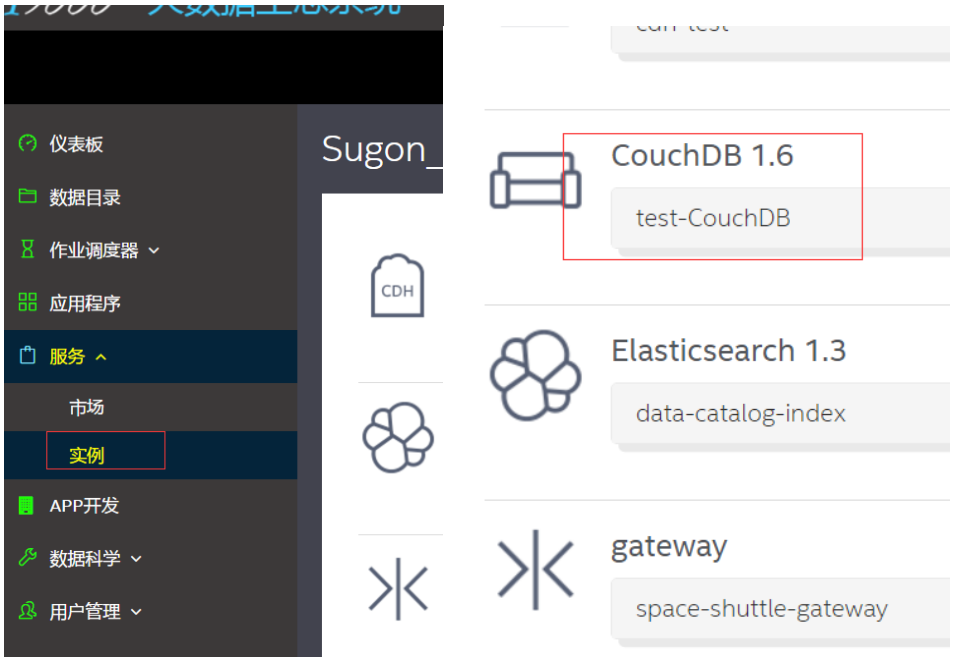
使用方法：

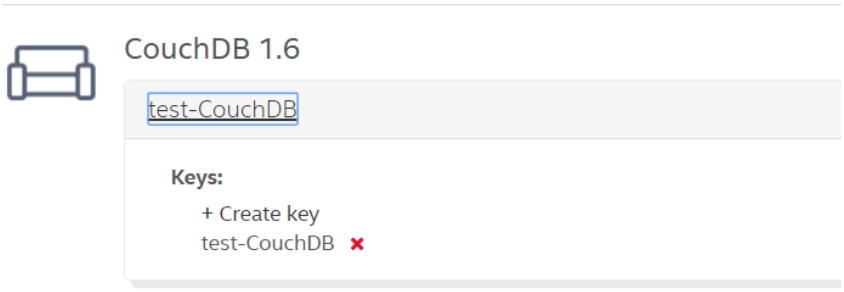
1. 在 i9000 平台上选择 服务>市场，找到 CouchDB，点击 CouchDB，创建一个新的实例。



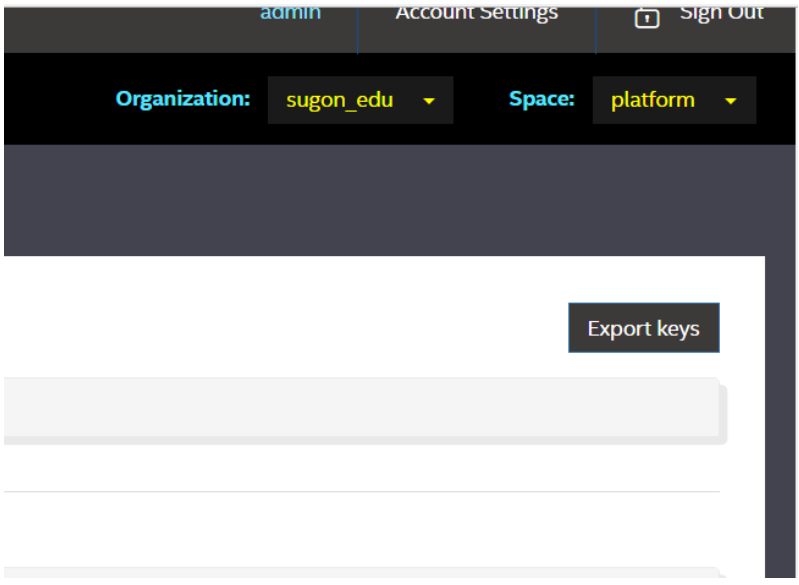


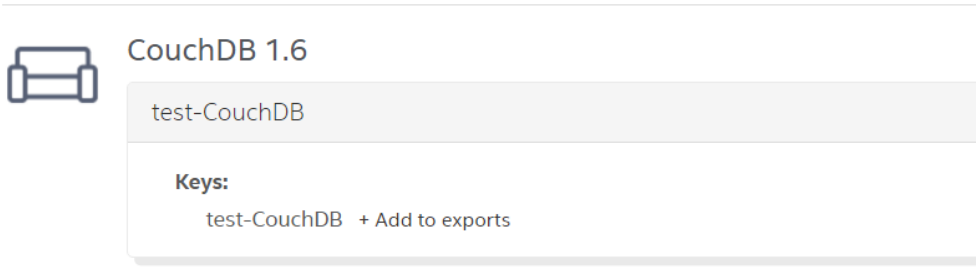
2. 在 服务>实例 里面找到 CouchDB，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 CouchDB 的连接信息。

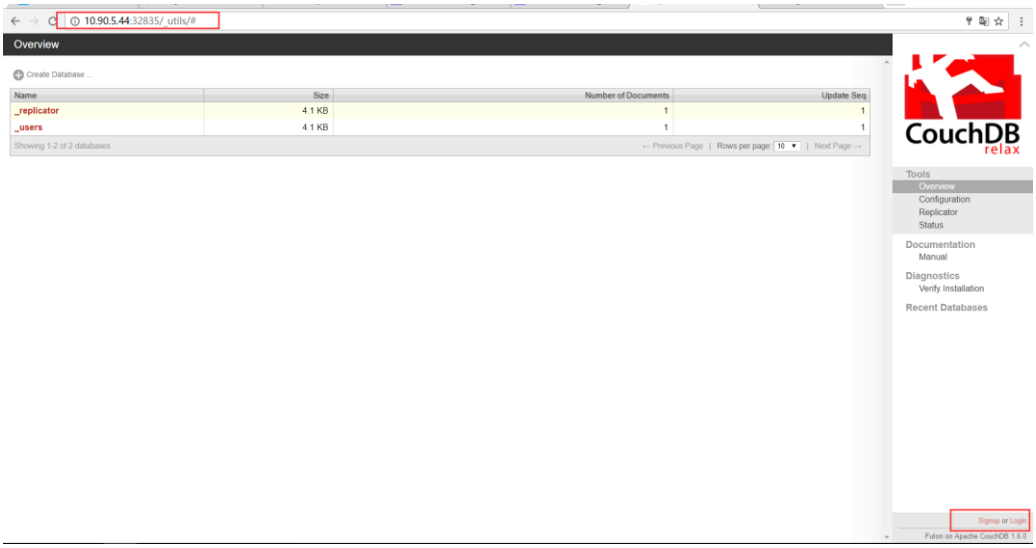




exported Keys:


```
{
  "couchdb16": [
    {
      "label": "couchdb16",
      "name": "test-CouchDB",
      "plan": "free",
      "tags": [
        "couchdb16",
        "couchdb",
        "document"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "5984/tcp": "32835"
        },
        "port": "32835",
        "username": "s91wvwtf0sc9rlh6",
        "password": "h49ukfgp3d8s3wn6",
        "dbname": "2a408xle2hf8oqa6",
        "uri": "couchdb://s91wvwtf0sc9rlh6:h49ukfgp3d8s3wn6@10.0.4.4:32835/2a408xle2hf8oqa6"
      }
    }
  ]
}
```


4. 在浏览器里输入 hostname:port/_utils 进行访问，可以看到界面化的 CouchDB,在右下角点击 Login，输入 username 和 password 可以登录。



Elasticsearch 概述

ElasticSearch 是一个基于 Lucene 的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎，基于 RESTful web 接口。Elasticsearch 是用 Java 开发的，并作为 Apache 许可条款下的开放源码发布，是当前流行的企业级搜索引擎。设计用于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便。

我们建立一个网站或应用程序，并要添加搜索功能，令我们受打击的是：搜索工作是很难的。我们希望我们的搜索解决方案要快，我们希望有一个零配置和一个完全免费的搜索模式，我们希望能够简单地使用 JSON 通过 HTTP 的索引数据，我们希望我们的搜索服务器始终可用，我们希望能够一台开始并扩展到数百，我们要实时搜索，我们要简单的多租户，我们希望建立一个云的解决方案。Elasticsearch 旨在解决所有这些问题和更多的问题。

Elasticsearch 基础概念

接近实时（NRT）

Elasticsearch 是一个接近实时的搜索平台。这意味着，从索引一个文档直到这个文档能够被搜索到有一个轻微的延迟（通常是 1 秒）。

集群（cluster）

一个集群就是由一个或多个节点组织在一起，它们共同持有你整个的数据，并一起提供索引和搜索功能。一个集群由一个唯一的名字标识，这个名字默认就是“elasticsearch”。这个名字是重要的，因为一个节点只能通过指定某个集群的名字，来加入这个集群。在产品环境中显式地设定这个名字是一个好习惯，但是使用默认值来进行测试/开发也是不错的。

节点（node）

一个节点是你集群中的一个服务器，作为集群的一部分，它存储你的数据，参与集群的索引和搜索功能。和集群类似，一个节点也是由一个名字来标识的，默认情况下，这个名字是一个随机的漫威漫画角色的名字，这个名字会在启动的时候赋予节点。这个名字对于管理工作来说挺重要的，因为在这个管理过程中，你会去确定网络中的哪些服务器对应于Elasticsearch 集群中的哪些节点。

一个节点可以通过配置集群名称的方式来加入一个指定的集群。默认情况下，每个节点都会被安排加入到一个叫做“elasticsearch”的集群中，这意味着，如果你在你的网络中启动了若干个节点，并假定它们能够相互发现彼此，它们将会自动地形成并加入到一个叫做“elasticsearch”的集群中。

在一个集群里，只要你想，可以拥有任意多个节点。而且，如果当前你的网络中没有运行任何 Elasticsearch 节点，这时启动一个节点，会默认创建并加入一个叫做“elasticsearch”的集群。

索引（index）

一个索引就是一个拥有几分相似特征的文档的集合。比如说，你可以有一个客户数据的索引，另一个产品目录的索引，还有一个订单数据的索引。一个索引由一个名字来标识（必须全部是小写字母的），并且当我们要对对应于这个索引中的文档进行索引、搜索、更新和删除的时候，都要使用到这个名字。在一个集群中，如果你想，可以定义任意多的索引。

类型（type）

在一个索引中，你可以定义一种或多种类型。一个类型是你的索引的一个逻辑上的分类/分区，其语义完全由你来定。通常，会为具有一组共同字段的文档定义一个类型。比如说，

我们假设你运营一个博客平台并且将你所有的数据存储到一个索引中。在这个索引中，你可以为用户数据定义一个类型，为博客数据定义另一个类型，当然，也可以为评论数据定义另一个类型。

文档（document）

一个文档是一个可被索引的基础信息单元。比如，你可以拥有某一个客户的文档，某一个产品的一个文档，当然，也可以拥有某个订单的一个文档。文档以 JSON（Javascript Object Notation）格式来表示，而 JSON 是一个到处存在的互联网数据交互格式。

在一个 index/type 里面，只要你想，你可以存储任意多的文档。注意，尽管一个文档，物理上存在于一个索引之中，文档必须被索引/赋予一个索引的 type。

分片和复制（shards & replicas）

一个索引可以存储超出单个结点硬件限制的大量数据。比如，一个具有 10 亿文档的索引占据 1TB 的磁盘空间，而任一节点都没有这样大的磁盘空间；或者单个节点处理搜索请求，响应太慢。

为了解决这个问题，Elasticsearch 提供了将索引划分成多份的能力，这些份就叫做分片。当你创建一个索引的时候，你可以指定你想要的分片的数量。每个分片本身也是一个功能完善并且独立的“索引”，这个“索引”可以被放置到集群中的任何节点上。

分片之所以重要，主要有两方面的原因：

- 允许你水平分割/扩展你的内容容量

- 允许你在分片（潜在地，位于多个节点上）之上进行分布式的、并行的操作，进而提

高性能/吞吐量

至于一个分片怎样分布，它的文档怎样聚合回搜索请求，是完全由 Elasticsearch 管理的，对于作为用户的你来说，这些都是透明的。

在一个网络/云的环境里，失败随时都可能发生，在某个分片/节点不知怎么的就处于离线状态，或者由于任何原因消失了，这种情况下，有一个故障转移机制是非常有用并且是强烈推荐的。为此目的，Elasticsearch 允许你创建分片的一份或多份拷贝，这些拷贝叫做复制分片，或者直接叫复制。

复制之所以重要，有两个主要原因：

- 在分片/节点失败的情况下，提供了高可用性。因为这个原因，注意到复制分片从不与原/主要（original/primary）分片置于同一节点上是非常重要的。

- 扩展你的搜索量/吞吐量，因为搜索可以在所有的复制上并行运行

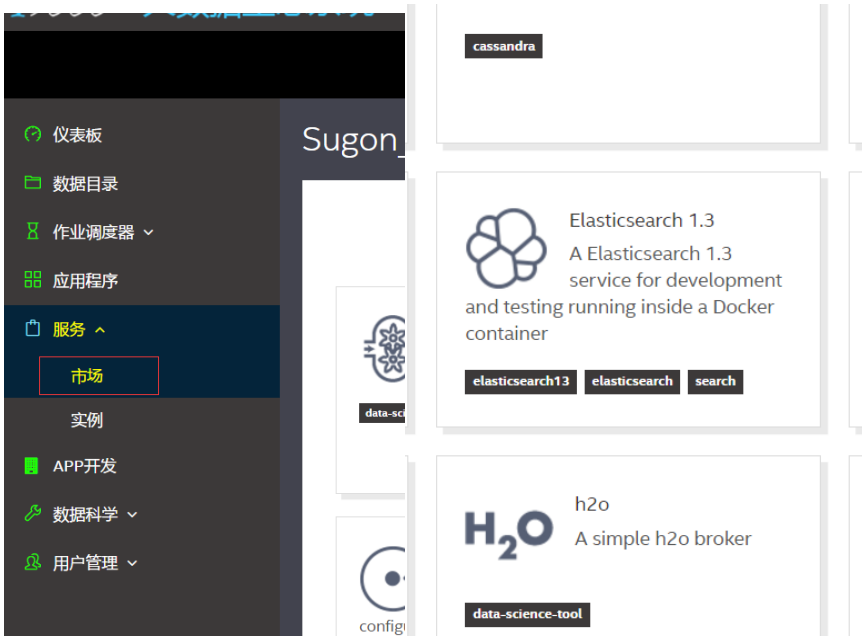
总之，每个索引可以被分成多个分片。一个索引也可以被复制 0 次（意思是没有复制）或多次。一旦复制了，每个索引就有了主分片（作为复制源的原来的分片）和复制分片（主分片的拷贝）之别。分片和复制的数量可以在索引创建的时候指定。在索引创建之后，你可以在任何时候动态地改变复制的数量，但是你事后不能改变分片的数量。

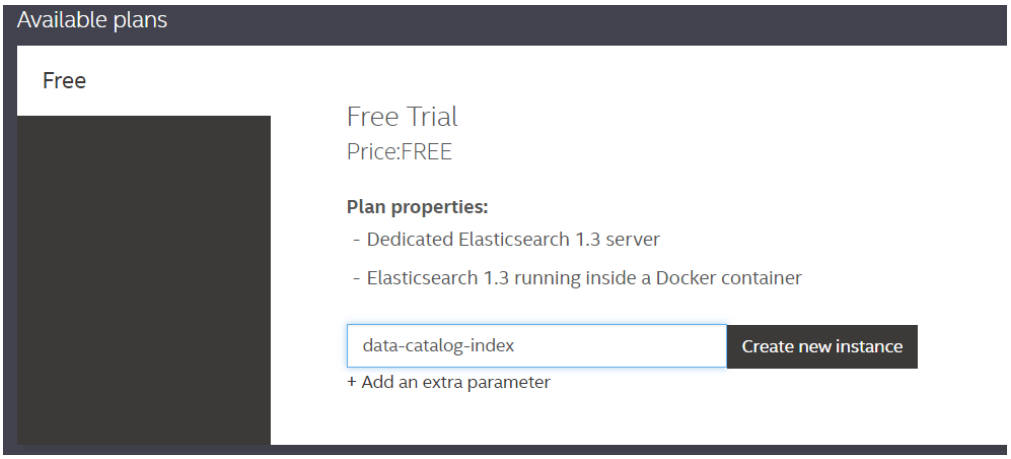
默认情况下，Elasticsearch 中的每个索引被分片 5 个主分片和 1 个复制，这意味着，如果你的集群中至少有两个节点，你的索引将会有 5 个主分片和另外 5 个复制分片（1 个完全拷贝），这样的话每个索引总共就有 10 个分片。

使用方法：

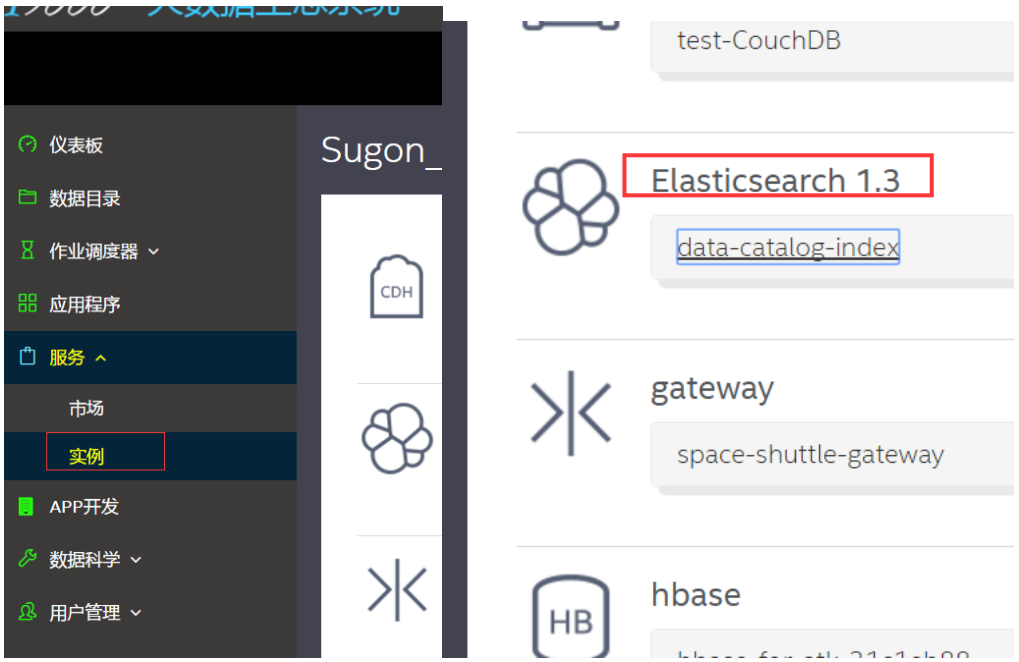
1. 在 i9000 平台上选择 服务>市场，找到 Elasticsearch 1.3，点击 Elasticsearch 1.3，

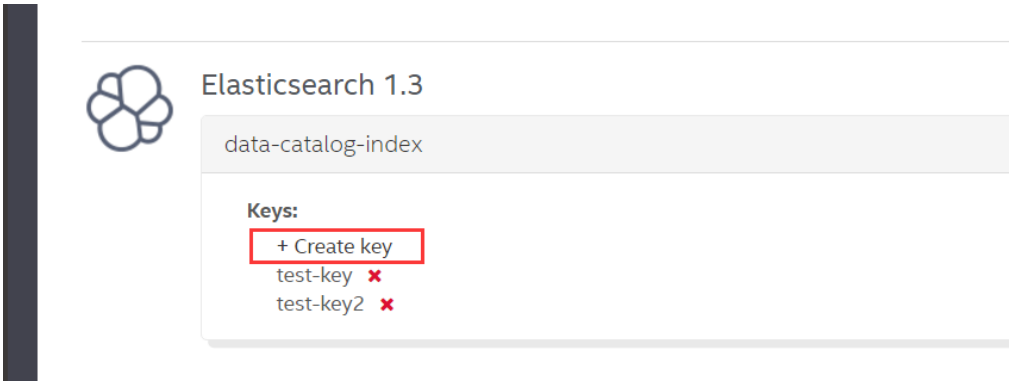
创建一个新的实例。



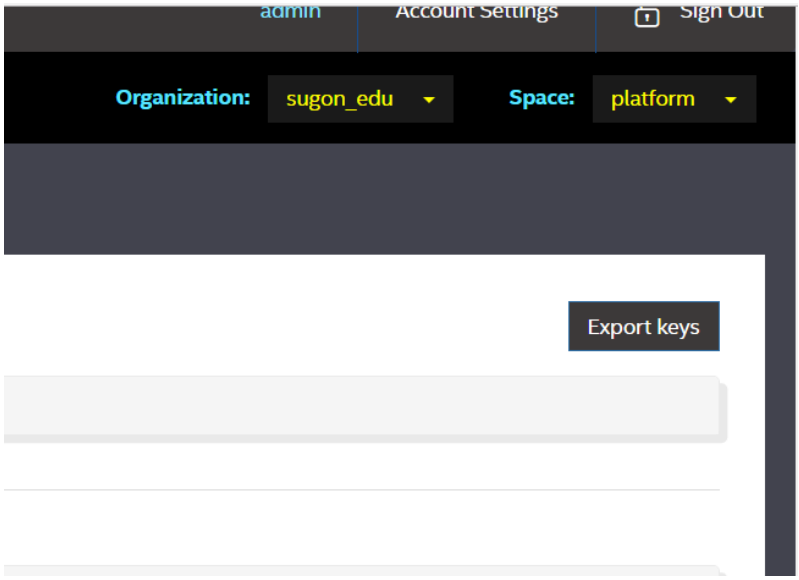


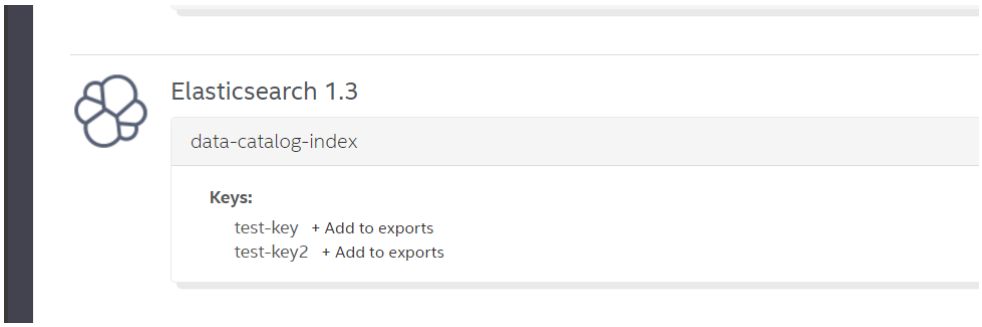
2. 在 服务>实例 里面找到 Elasticsearch 1.3 ,找到刚刚创建的实例 ,点击 ,创建新的 key。



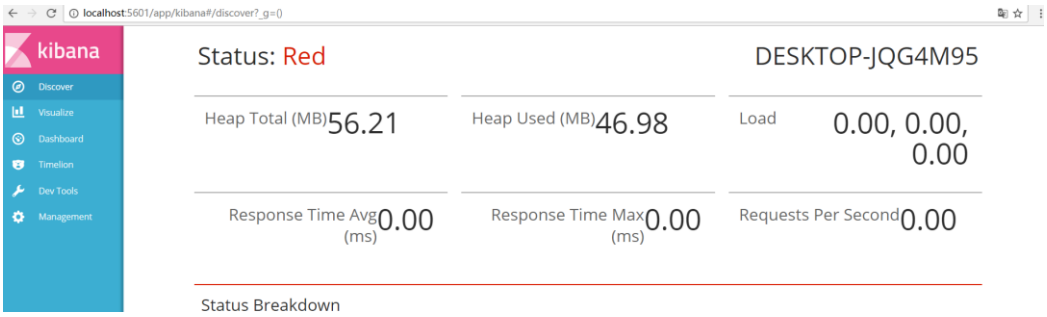


3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Elasticsearch 1.3 的连接信息。





4. 在 Elasticsearch 可视化工具里（如 Kibana）使用 hostname 和 port 即可连接到新创建的 Elasticsearch 实例，其中 9200 映射的端口为 Http 传输监听定制端口，9300 映射的端口为配置节点之间交互的端口。


localhost:5601/app/kibana#/discover?_g=()
kibana
Discover
Visualize
Dashboard
Timelion
Dev Tools
Management
Status: Red
DESKTOP-JQG4M95
Heap Total (MB) 56.21
Heap Used (MB) 46.98
Load 0.00, 0.00, 0.00
Response Time Avg (ms) 0.00
Response Time Max (ms) 0.00
Requests Per Second 0.00
Status Breakdown

Etcd 概述

etcd 是一个高可用的键值存储系统，主要用于共享配置和服务发现。etcd 是由 CoreOS 开发并维护的，灵感来自于 ZooKeeper 和 Doozer，它使用 Go 语言编写，并通过 Raft 一致性算法处理日志复制以保证强一致性。Raft 是一个来自 Stanford 的新的一致性算法，适用于分布式系统的日志复制，Raft 通过选举的方式来实现一致性，在 Raft 中，任何一个节点都可能成为 Leader。Google 的容器集群管理系统 Kubernetes、开源 PaaS 平台 Cloud Foundry 和 CoreOS 的 Fleet 都广泛使用了 etcd。

在分布式系统中，如何管理节点间的状态一直是一个难题，etcd 像是专门为集群环境的服务发现和注册而设计，它提供了数据 TTL 失效、数据改变监视、多值、目录监听、分布式锁原子操作等功能，可以方便的跟踪并管理集群节点的状态。etcd 目前的版本是 0.4.5，虽然未发布 1.0 版本（今年会发布），但其已经使用在多个生产系统中，可见其火热程度。etcd 的特性如下：

简单: curl 可访问的用户的 API（HTTP+JSON）

安全: 可选的 SSL 客户端证书认证

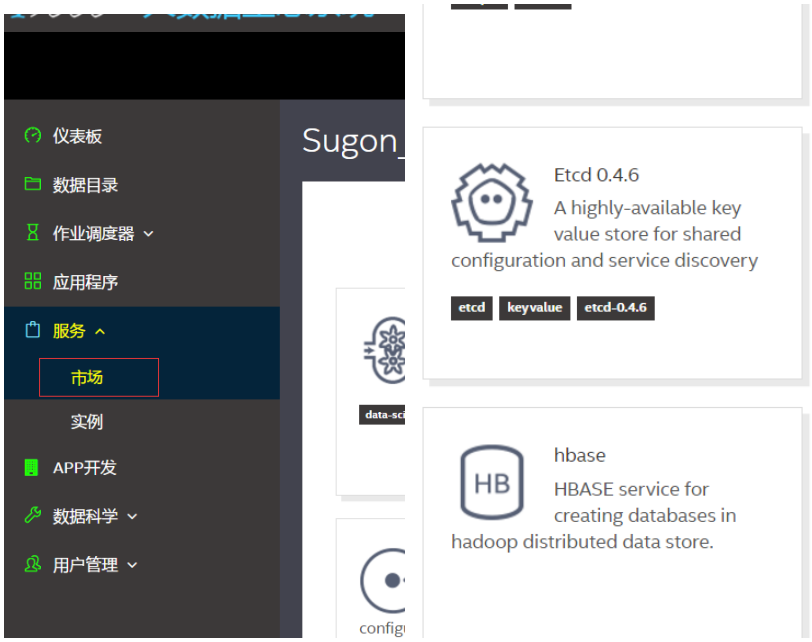
快速: 单实例每秒 1000 次写操作

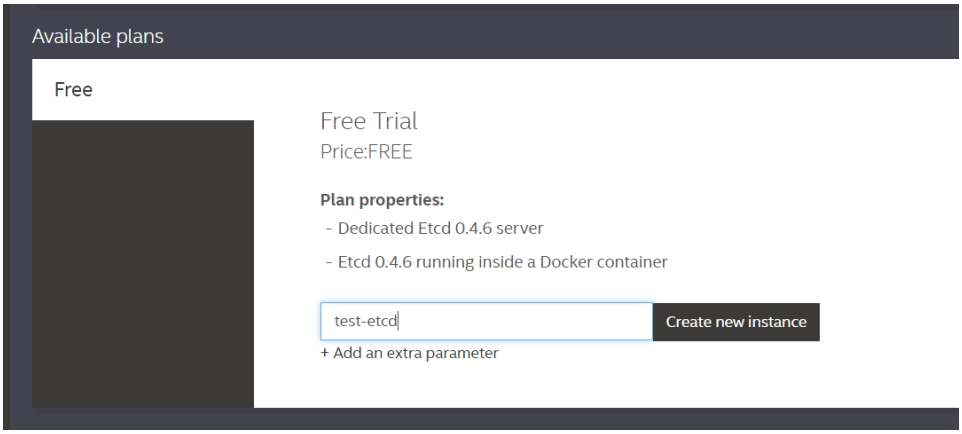
可靠: 使用 Raft 保证一致性

etcd 是 CoreOS 的核心组件，负责节点间的服务发现和配置共享，运行在 CoreOS 中的应用可以通过 etcd 读取或者写入数据。虽然 etcd 是为 CoreOS 而设计，但其可以运行在多个平台上，包括 OS X、Linux、BSD。

使用方法：

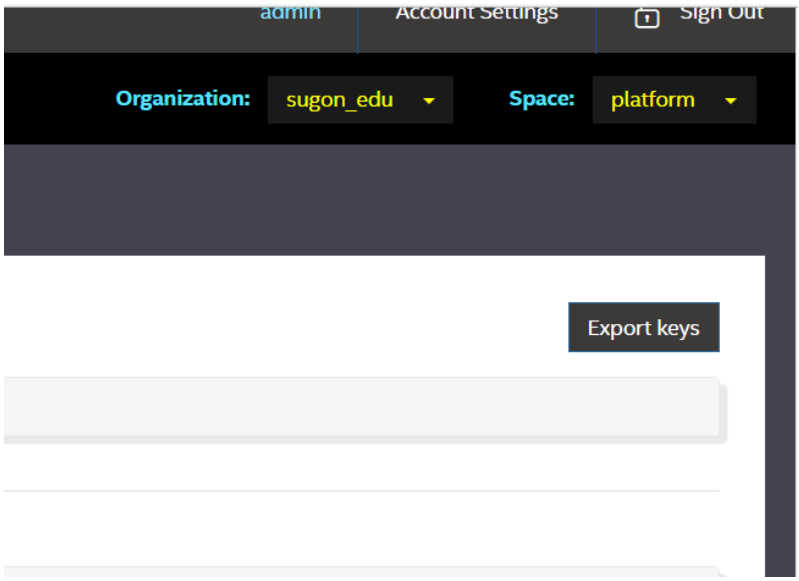
1. 在 i9000 平台上选择 服务>市场，找到 Etcd，点击 Etcd，创建一个新的实例

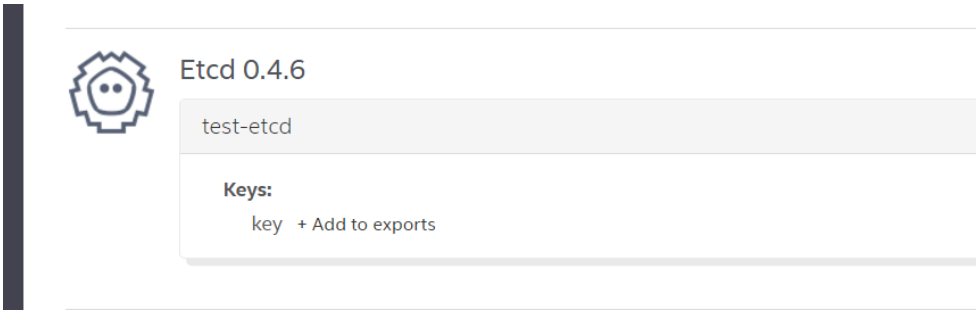




2. 在 服务>实例 里面找到 Etcd，找到刚刚创建的实例，点击，创建新的 key。

3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Etcd 的配置信息。





4. 使用 hostname 和 port 即可使用新创建的 Etcd 实例，其中 4001 映射的端口用来和 client 进行通讯，7001 映射的端口是服务端和服务端之间进行通讯的端口。

Gateway 概述

Gateway 是从 WebSocket 到 Kafka 的简单桥梁，它暴露了将所有入站消息发布到可配置消息后端（Apache Kafka 队列）的 WebSocket 接口，它与任何基于 Websocket 的发布商兼容。

实现方式：

•JSON 格式的数据，不需要映射

•基于 Token 的客户端认证（OAuth 2.0）

•支持多个后端（默认为 Apache Kafka）

•可配置服务器（端口，路径，服务等）

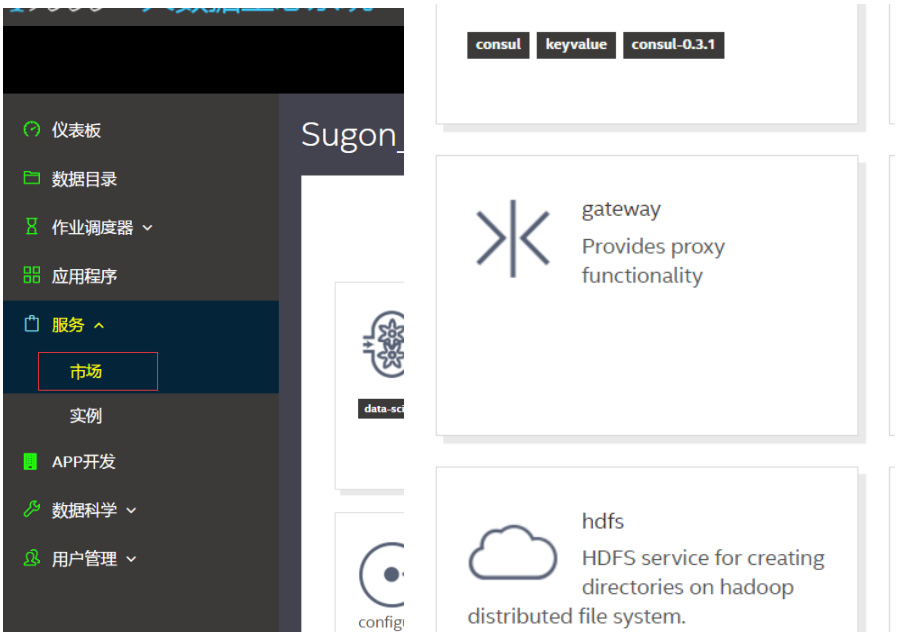
功能：

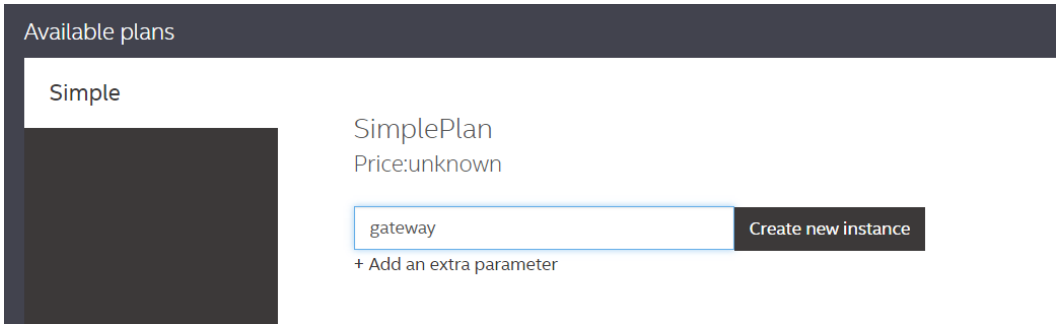
•客户级授权

•动态后端配置

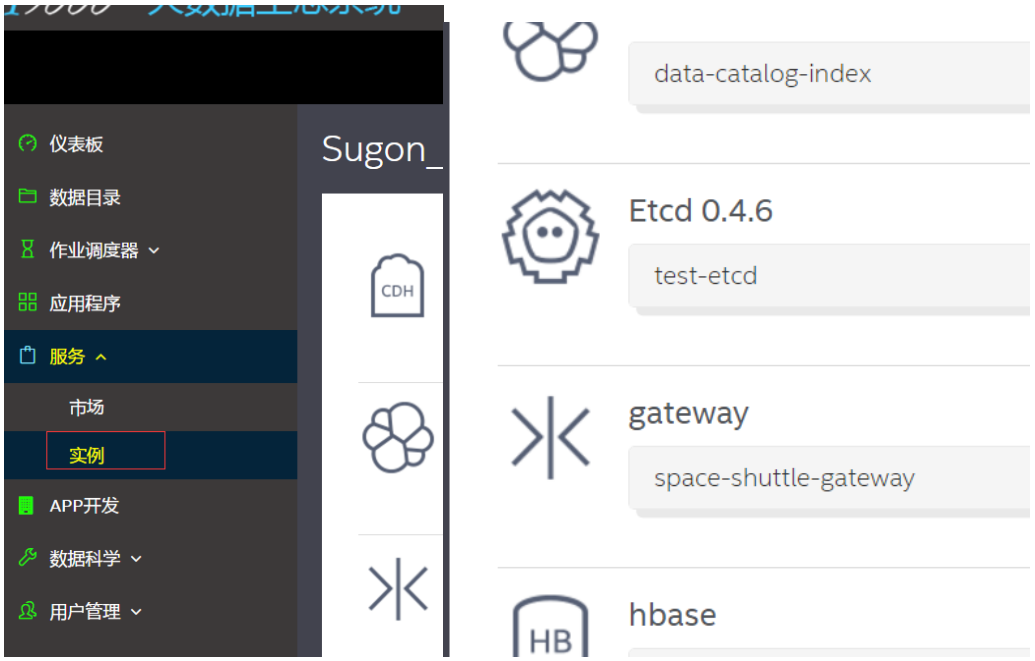
使用方法：

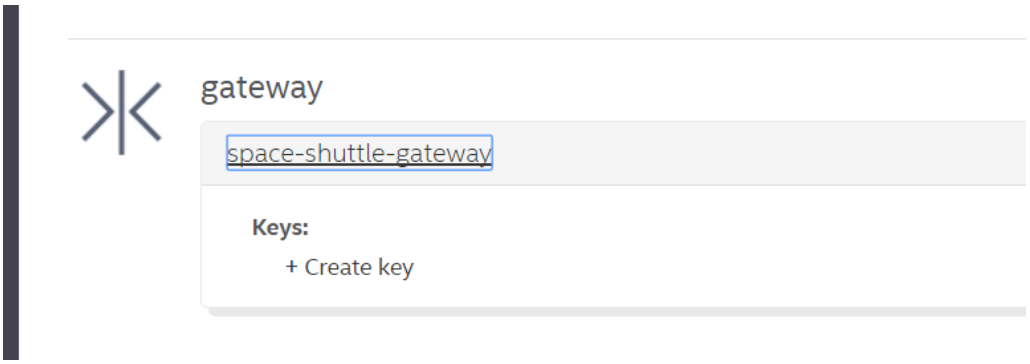
1. 在 i9000 平台上选择 服务>市场，找到 Gateway，点击 Gateway，创建一个新的实例。



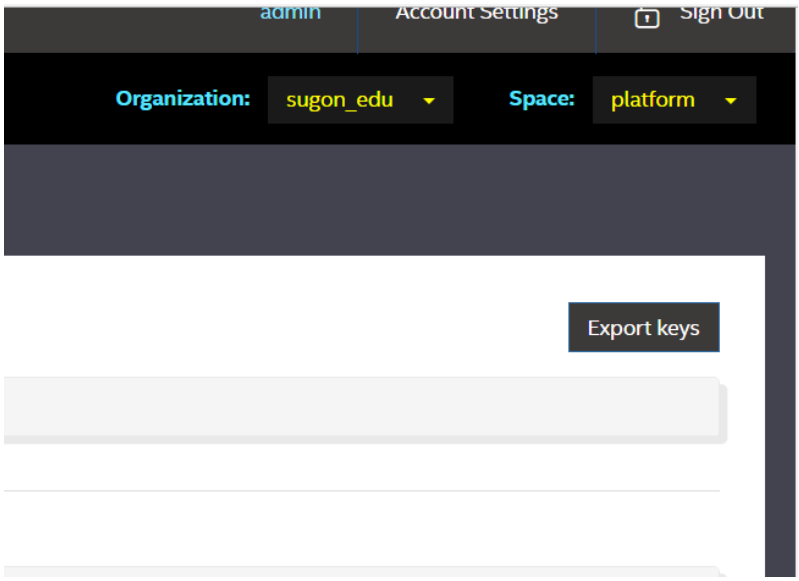


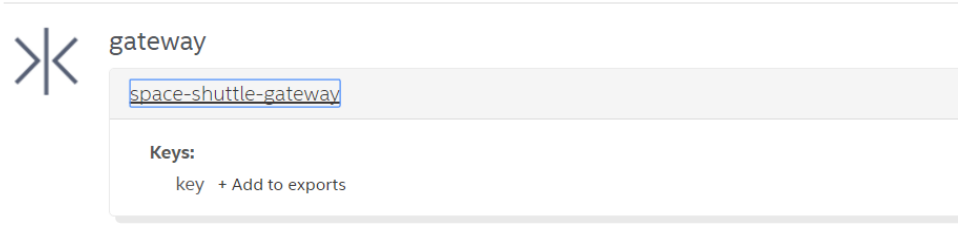
2. 在 服务>实例 里面找到 Gateway，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Gateway 的配置信息。





exported Keys:


```
{
  "gateway": [
    {
      "label": "gateway",
      "name": "space-shuttle-gateway",
      "plan": "Simple",
      "tags": [],
      "credentials": {
        "url": "space-shuttle-gateway-669afedb.i9000.com"
      }
    }
  ]
}
```


4. 这里可以看到新建 Gateway 的 url ,用此 url 可以在应用程序中绑定此 Gateway 实例。

```
def parse_arguments():
    parser = argparse.ArgumentParser(
        description='Deployment script for Space Shuttle client')
    parser.add_argument('--gateway-url', type=str,
                        help='gateway api url, '
                        'e.g. gateway-479613d7.i9000.com')
    parser.add_argument('--use-https', dest='https', action='store_true',
                        help='set of flag cause use of `https_proxy` env '
                             'instead of default `http_proxy`')
    return parser.parse_args()
```

GearPump Dashboard 概述

GearPump Dashboard 是管理 GearPump 服务的仪表盘。Gearpump 是一个基于 Akka Actor 的轻量级的实时流计算引擎。今天的流平台需要能处理来自各种移动端和物联网设备的海量数据，系统要能不间断的提供服务，对数据的处理要能做到不丢失不重复，对各种软硬件错误能平滑处理，对用户的输入要能实时响应。除了这些系统层面的需求外，用户层面的接口还要能做到丰富而灵活，一方面，平台要提供足够丰富的基础设施，能最简化应用程序的编写；另一方面，这个平台应提供具有表现力的编程 API，让用户能灵活表达各种计算，并且整个系统可以定制，允许用户选择调度策略和部署环境，允许用户在不同的指标间做折中取舍，满足特定的需求。Akka Actor 提供了通信、并发、隔离、容错的基础设施，Gearpump 通过把抽象层次提升到 Actor 这一层，屏蔽了底层的细节，专注于流处理需求的本身，能更简单而又高效的解决上述问题。

Apache Gearpump 能解决实时计算，实时反馈，实时机器学习和数据分析等各种大数据的实时问题，适用于金融，物联网，企业云，医疗等各种应用场景。它有以下优点：

支持 event time 的消息处理，支持 out of order 消息处理.

作者常年做性能优化，Gearpump 有极高的性能，一秒钟能处理上千万消息

毫秒级的延时，同时支持 exactly-once 消息处理

非常易用的 Dashboard.

支持企业内部的 Cgroup 资源隔离。

支持 Akka Stream API，并通过 Akka Stream 可以和各种提供 Reactive stream 接口

数据系统相连。

Binary 兼容大部分 Storm 的应用 jar。

支持业务计算图的在线替换和升级，升级过程中没有消息丢失，没有 service down time。

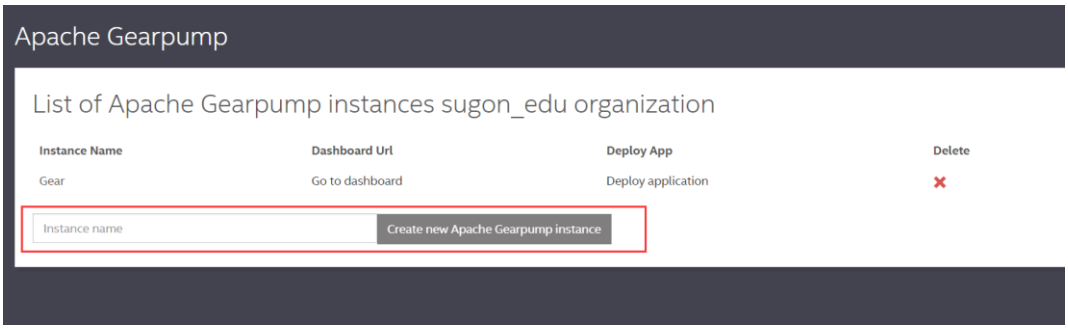
极简的用户 API。

部分支持 Apache Beam 的 API。

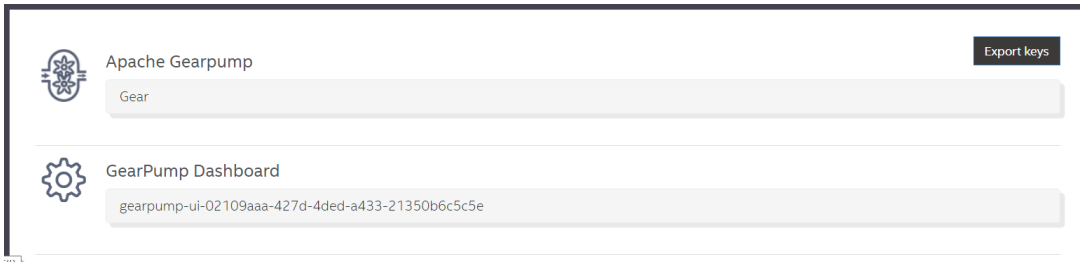
使用方法：

GearPump Dashboard 要和平台上的 Apache Gearpump 绑定使用，当创建一个 Apache Gearpump 的实例的时候平台会自动创建一个 Gearpump Dashboard 与之绑定在一起。

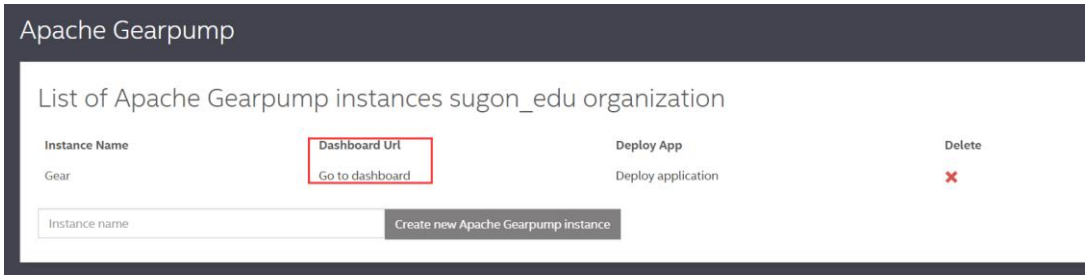
1. 在 i9000 平台导航栏里选择 数据科学>Apache Gearpump，在打开的页面创建一个新的实例。

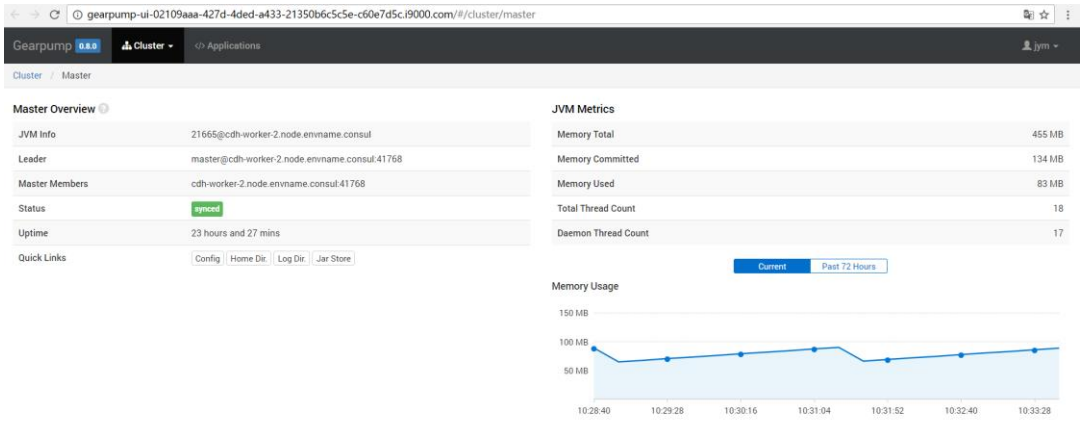


2. 创建完成后，在导航栏 服务>实例 里可以看到同时创建了 Apache Gearpump 和 Gearpump Dashboard 两个实例。



3. 回到 数据科学>Apache Gearpump 页面，点击新创建实例的 Dashboard Url 可以使用 Apache Gearpump。





H2O 概述

H2O 是一种分布式的内存处理引擎用于机器学习，它拥有一个令人印象深刻的数组的算法。早期版本仅仅支持 R 语言，3.0 版本开始支持 Python 和 Java 语言，同时它也可以作为 Spark 在后端的执行引擎。

使用 H2O 的最佳方式是把它作为 R 环境的一个大内存扩展，R 环境并不直接作用于大的数据集，而是通过扩展通讯协议例如 REST API 与 H2O 集群通讯，H2O 来处理大量的数据工作。

几个有用的 R 扩展包，如 ddply 已经被打包，允许你在处理大规模数据集时，打破本地机器上内存容量的限制。你可以在 EC2 上运行 H2O，或者 Hadoop 集群/YARN 集群，或者 Docker 容器。用苏打水(Spark+ H2O)你可以访问在集群上并行的访问 Spark RDDS，在数据帧被 Spark 处理后。再传递给一个 H2O 的机器学习算法。

H2O 能够让 Hadoop 做数学，H2O 是基于大数据的统计分析机器学习和数学库包，让用户基于核心的数学积木搭建应用块代码，采取类似 R 语言 Excel 或 JSON 等熟悉接口，使的 BigData 爱好者和专家可以利用一系列简单的先进算法对数据集进行探索，建模和评估。数据收集是很容易，但是决策是很难的。H2O 使得能用更快更好的预测模型源实现快速和方便地数据的挖掘。H2O 愿意将在线评分和建模融合在一个单一平台上。

H2O 具有其他机器学习平台当前所没有的功能：

最佳开源技术 – 享受开源技术大数据科学所带来的自由。H2O 通过最流行的开源产品 ApacheTM Hadoop 和 SparkTM 使客户更加灵活方便地解决最有挑战性数据问题。

易于使用的 WebUI 和熟悉的接口 – 使用 H2O 的基于 Web 的用户界面或熟悉的编程

环境（例如 R、Java、Scala、Python、JSON）可以快速设置并入门。

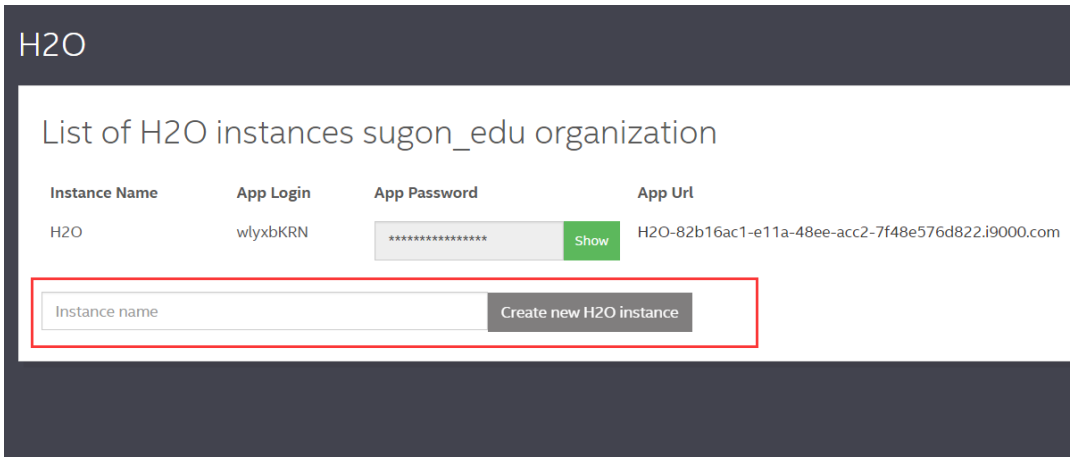
支持所有通用数据库和文件类型 – 对微软 Excel、R Studio、Tableau 等软件内的大数据可以轻松浏览和建模。能够连接 HDFS、S3、SQL 和 NoSQL 数据源。 可在任何地方安装和部署。

海量大数据分析 – 在全部数据集而不是小样本上训练模型，通过 H2O 快速 in-memory 分布平行处理实时迭代和开发模型。

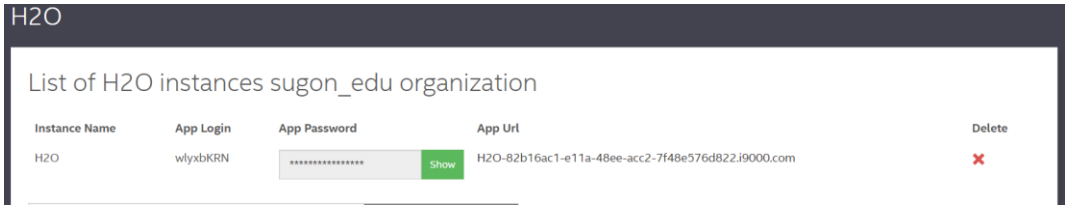
实时数据评分 – 为了精确预测，使用 Nanofast 评分引擎可在任何环境以纳秒级对模型进行数据评分。它比当前市场上最接近的技术（说的是谁？）在评分和预测上快十倍。

使用方法：

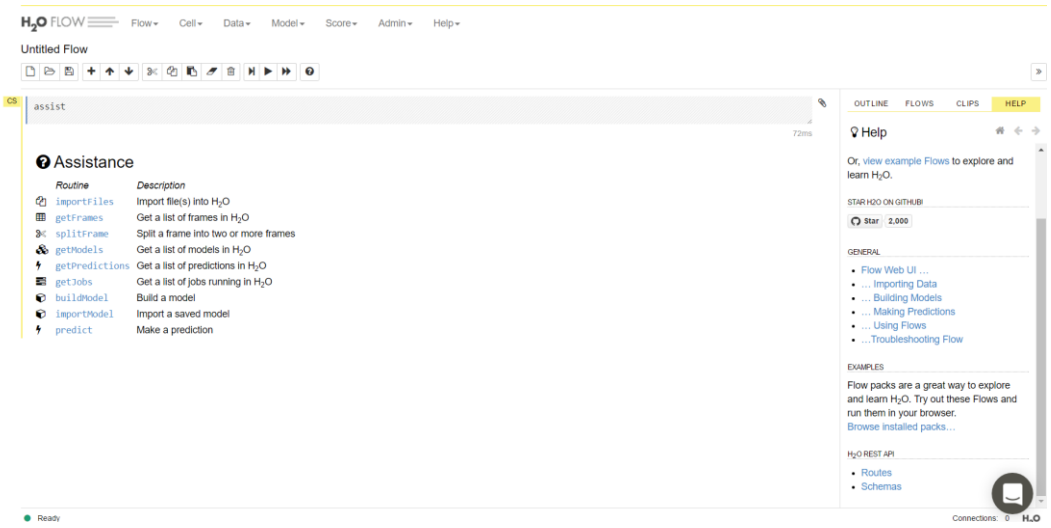
1. 在 i9000 平台上选择 数据科学，找到 H2O，点击 H2O，创建一个新的实例。



2. 创建过程可能需要 1-2 分钟，完成后刷新页面即可看到刚刚创建的实例。



3. 点击绿色的“show”按钮可以看到 App Password 的明文，点击右侧 App Url，输入 App Login 和 App Password 即可访问 H2O。



右侧 Help 提供了 H2O 的使用帮助，可链接到官网。

Hbase 概述

HBase 是一个分布式的、面向列的开源数据库，该技术来源于 Fay Chang 所撰写的 Google 论文“Bigtable：一个结构化数据的分布式存储系统”。就像 Bigtable 利用了 Google 文件系统（File System）所提供的分布式数据存储一样，HBase 在 Hadoop 之上提供了类似于 Bigtable 的能力。HBase 是 Apache 的 Hadoop 项目的子项目。HBase 不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据库。另一个不同的是 HBase 基于列的而不是基于行的模式。

结构介绍：

HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。与 FUJITSU Cliq 等商用大数据产品不同，HBase 是 Google Bigtable 的开源实现，类似 Google Bigtable 利用 GFS 作为其文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 Bigtable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，HBase 利用 Zookeeper 作为对应。

上图描述 Hadoop EcoSystem 中的各层系统。其中,HBase 位于结构化存储层，Hadoop HDFS 为 HBase 提供了高可靠性的底层存储支持，Hadoop MapReduce 为 HBase 提供了高性能的计算能力，Zookeeper 为 HBase 提供了稳定服务和 failover 机制。

此外，Pig 和 Hive 还为 HBase 提供了高层语言支持，使得在 HBase 上进行数据统计处理变的非常简单。 Sqoop 则为 HBase 提供了方便的 RDBMS 数据导入功能，使得传统数据库数据向 HBase 中迁移变的非常方便。

访问接口：

1. Native Java API，最常规和高效的访问方式，适合 Hadoop MapReduce Job 并行批处理 HBase 表数据

2. HBase Shell，HBase 的命令行工具，最简单的接口，适合 HBase 管理使用

3. Thrift Gateway，利用 Thrift 序列化技术，支持 C++，PHP，Python 等多种语言，适合其他异构系统在线访问 HBase 表数据

4. REST Gateway，支持 REST 风格的 Http API 访问 HBase，解除了语言限制

5. Pig，可以使用 Pig Latin 流式编程语言来操作 HBase 中的数据，和 Hive 类似，本质最终也是编译成 MapReduce Job 来处理 HBase 表数据，适合做数据统计

6. Hive ,当前 Hive 的 Release 版本尚没有加入对 HBase 的支持 ,但在下一个版本 Hive 0.7.0 中将会支持 HBase，可以使用类似 SQL 语言来访问 HBase

HBase 数据模型 Table & Column Family

Row Key	Timestamp	Column Family	
URI	Parser		
r1	t3	url=http://	title=
t2	host=com		
t1			
r2	t5	url=http://	content=每天...
t4	host=com		

Row Key: 行键，Table 的主键，Table 中的记录默认按照 Row Key 升序排序。

Timestamp:时间戳，每次数据操作对应的时间戳，可以看作是数据的 version number。

Column Family：列簇，Table 在水平方向有一个或者多个 Column Family 组成，一个 Column Family 中可以由任意多个 Column 组成，即 Column Family 支持动态扩展，无需预先定义 Column 的数量以及类型，所有 Column 均以二进制格式存储，用户需要自行进行类型转换。

Table & Region

当 Table 随着记录数不断增加而变大后，会逐渐分裂成多份 splits，成为 regions，一个 region 由[startkey,endkey)表示，不同的 region 会被 Master 分配给相应的 RegionServer 进行管理：

-ROOT- && .META. Table

HBase 中有两张特殊的 Table，-ROOT-和.META.

.META.：记录了用户表的 Region 信息，.META.可以有多个 regoin

-ROOT-：记录了.META.表的 Region 信息，-ROOT-只有一个 region

Ø Zookeeper 中记录了-ROOT-表的 location

Client 访问用户数据之前需要首先访问 zookeeper，然后访问-ROOT-表，接着访问.META.表，最后才能找到用户数据的位置去访问，中间需要多次网络操作，不过 client 端会做 cache 缓存。

HBase 系统架构

ClientHBase Client 使用 HBase 的 RPC 机制与 HMaster 和 HRegionServer 进行通信，对于管理类操作，Client 与 HMaster 进行 RPC；对于数据读写类操作，Client 与

HRegionServer 进行 RPC。

1 Zookeeper

Zookeeper Quorum 中除了存储了 -ROOT- 表的地址和 HMaster 的地址，HRegionServer 也会把自己以 Ephemeral 方式注册到 Zookeeper 中，使得 HMaster 可以随时感知到各个 HRegionServer 的健康状态。此外，Zookeeper 也避免了 HMaster 的单点问题，见下文描述

HMaster

HMaster 没有单点问题，HBase 中可以启动多个 HMaster，通过 Zookeeper 的 Master Election 机制保证总有一个 Master 运行，HMaster 在功能上主要负责 Table 和 Region 的管理工作：

1. 管理用户对 Table 的增、删、改、查操作

2. 管理 HRegionServer 的负载均衡，调整 Region 分布

3. 在 Region Split 后，负责新 Region 的分配

4. 在 HRegionServer 停机后，负责失效 HRegionServer 上的 Regions 迁移

HRegionServer

HRegionServer 主要负责响应用户 I/O 请求，向 HDFS 文件系统中读写数据，是 HBase 中最核心的模块。

HRegionServer 内部管理了一系列 HRegion 对象，每个 HRegion 对应了 Table 中的一个 Region，HRegion 中由多个 HStore 组成。每个 HStore 对应了 Table 中的一个 Column Family 的存储，可以看出每个 Column Family 其实就是一个集中的存储单元，因此最好将

具备共同 IO 特性的 column 放在一个 Column Family 中，这样最高效。

HStore 存储是 HBase 存储的核心了，其中由两部分组成，一部分是 MemStore，一部分是 StoreFiles。MemStore 是 Sorted Memory Buffer，用户写入的数据首先会放入 MemStore，当 MemStore 满了以后会 Flush 成一个 StoreFile（底层实现是 HFile），当 StoreFile 文件数量增长到一定阈值，会触发 Compact 合并操作，将多个 StoreFiles 合并成一个 StoreFile，合并过程中会进行版本合并和数据删除，因此可以看出 HBase 其实只有增加数据，所有的更新和删除操作都是在后续的 compact 过程中进行的，这使得用户的写操作只要进入内存中就可以立即返回，保证了 HBase I/O 的高性能。当 StoreFiles Compact 后，会逐步形成越来越大的 StoreFile，当单个 StoreFile 大小超过一定阈值后，会触发 Split 操作，同时把当前 Region Split 成 2 个 Region，父 Region 会下线，新 Split 出的 2 个孩子 Region 会被 HMaster 分配到相应的 HRegionServer 上，使得原先 1 个 Region 的压力得以分流到 2 个 Region 上。下图描述了 Compaction 和 Split 的过程：

在理解了上述 HStore 的基本原理后，还必须了解一下 HLog 的功能，因为上述的 HStore 在系统正常工作的前提下是没有问题的，但是在分布式系统环境中，无法避免系统出错或者宕机，因此一旦 HRegionServer 意外退出，MemStore 中的内存数据将会丢失，这就需要引入 HLog 了。每个 HRegionServer 中都有一个 HLog 对象，HLog 是一个实现 Write Ahead Log 的类，在每次用户操作写入 MemStore 的同时，也会写一份数据到 HLog 文件中（HLog 文件格式见后续），HLog 文件定期会滚动出新的，并删除旧的文件（已持久化到 StoreFile 中的数据）。当 HRegionServer 意外终止后，HMaster 会通过 Zookeeper 感知到，HMaster 首先会处理遗留的 HLog 文件，将其中不同 Region 的 Log 数据进行拆分，分别放到相应 region 的目录下，然后再将失效的 region 重新分配，领取 到这些 region

的 HRegionServer 在 Load Region 的过程中，会发现有历史 HLog 需要处理，因此会 Replay HLog 中的数据到 MemStore 中，然后 flush 到 StoreFiles，完成数据恢复。

存储格式

HBase 中的所有数据文件都存储在 Hadoop HDFS 文件系统上，主要包括上述提出的两种文件类型：

1. HFile， HBase 中 KeyValue 数据的存储格式，HFile 是 Hadoop 的二进制格式文件，实际上 StoreFile 就是对 HFile 做了轻量级包装，即 StoreFile 底层就是 HFile。

2. HLog File，HBase 中 WAL（Write Ahead Log） 的存储格式，物理上是 Hadoop 的 Sequence File。

HFile

首先 HFile 文件是不定长的，长度固定的只有其中的两块：Trailer 和 FileInfo。正如图中所示的，Trailer 中有指针指向其他数据块的起始点。File Info 中记录了文件的一些 Meta 信息，例如：AVG_KEY_LEN, AVG_VALUE_LEN, LAST_KEY, COMPARATOR, MAX_SEQ_ID_KEY 等。Data Index 和 Meta Index 块记录了每个 Data 块和 Meta 块的起始点。

Data Block 是 HBase I/O 的基本单元，为了提高效率，HRegionServer 中有基于 LRU 的 Block Cache 机制。每个 Data 块的大小可以在创建一个 Table 的时候通过参数指定，大号的 Block 有利于顺序 Scan，小号 Block 利于随机查询。每个 Data 块除了开头的 Magic 以外就是一个个 KeyValue 对拼接而成, Magic 内容就是一些随机数字，目的是防止数据损坏。后面会详细介绍每个 KeyValue 对的内部构造。

HFile 里面的每个 KeyValue 对就是一个简单的 byte 数组。但是这个 byte 数组里面包

含了很多项，并且有固定的结构。我们来看看里面的具体结构：

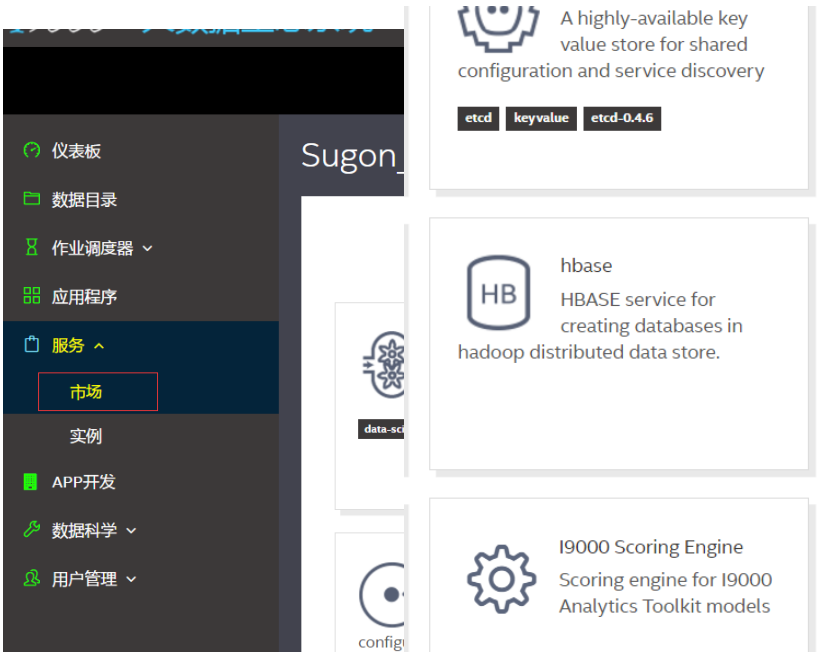
开始是两个固定长度的数值，分别表示 Key 的长度和 Value 的长度。紧接着是 Key，开始是固定长度的数值，表示 RowKey 的长度，紧接着是 RowKey，然后是固定长度的数值，表示 Family 的长度，然后是 Family，接着是 Qualifier，然后是两个固定长度的数值，表示 Time Stamp 和 Key Type（Put/Delete）。Value 部分没有这么复杂的结构，就是纯粹的二进制数据了。

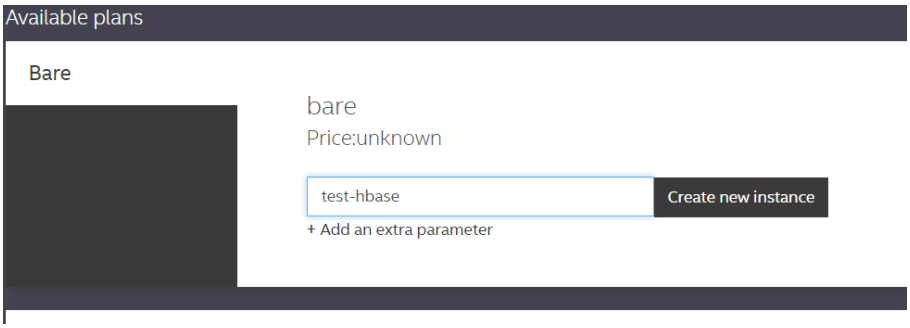
HLogFile

HLog 文件是一个普通的 Hadoop Sequence File，Sequence File 的 Key 是 HLogKey 对象，HLogKey 中记录了写入数据的归属信息，除了 table 和 region 名字外，同时还包括 sequence number 和 timestamp，timestamp 是“写入时间”，sequence number 的起始值为 0，或者是最近一次存入文件系统中 sequence number。

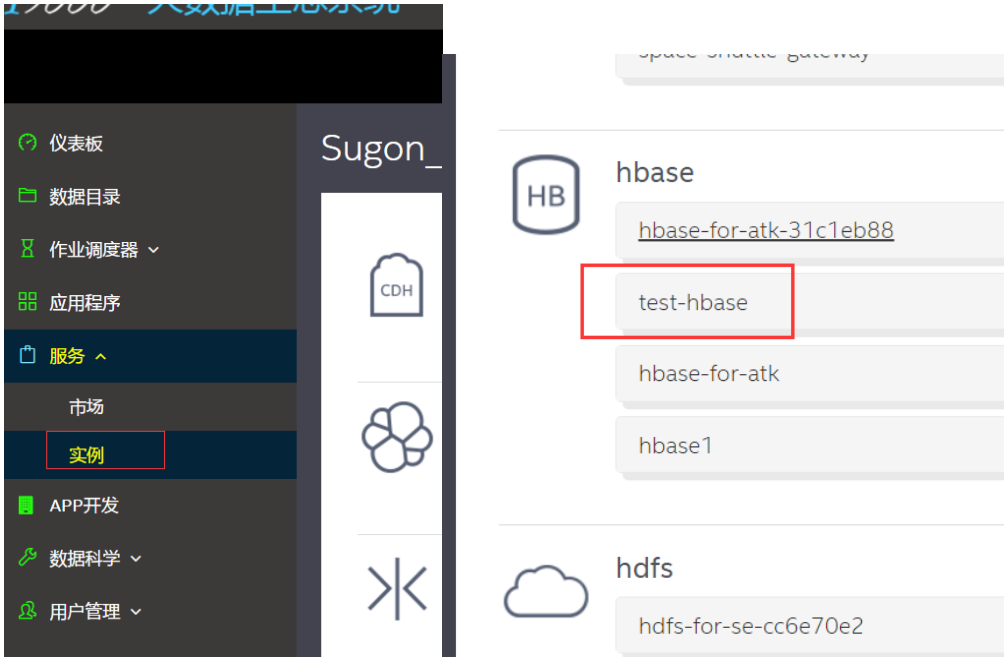
使用方法：

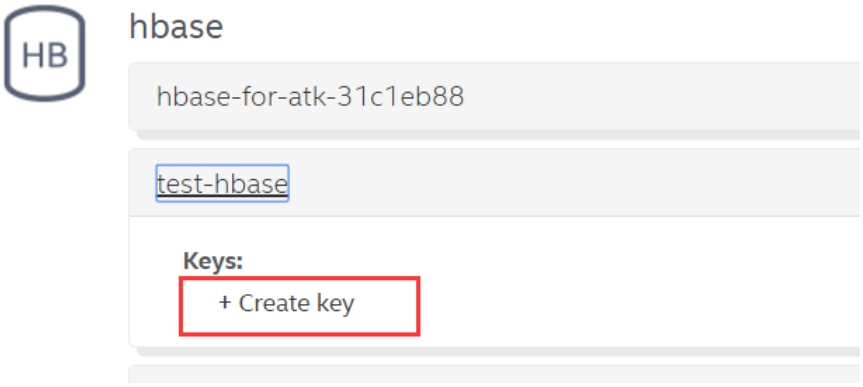
1. 在 i9000 平台上选择 服务>市场，找到 hbase，点击 hbase，创建一个新的实例。



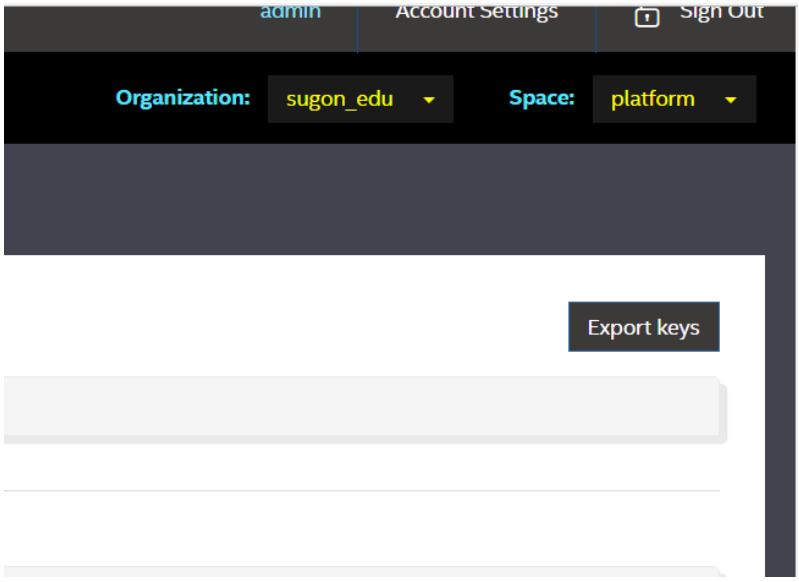


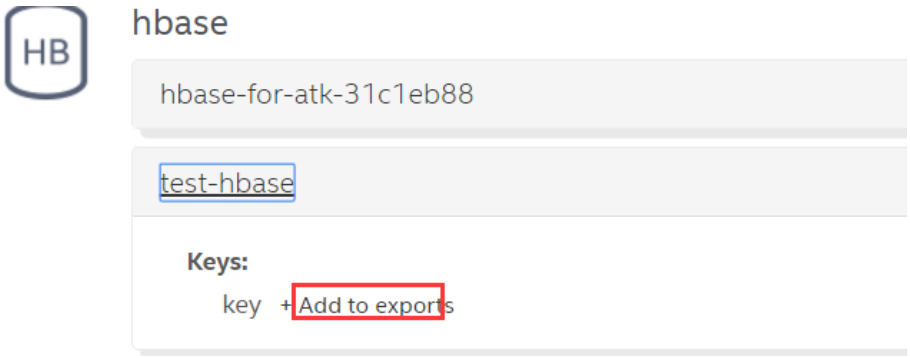
2. 在 服务>实例 里面找到 hbase，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 hbase 的连接信息。





```
    "hbase.client.primaryCallTimeout.get": "10",
    "hadoop.security.authorization": "false",
    "hbase.client.retries.number": "35",
    "hadoop.proxyuser.HTTP.hosts": "*",
    "hadoop.proxyuser.httpfs.groups": "*",
    "hbase.zookeeper.quorum": "cdh-master-0.node.envname.consul,cdh-master-1.node.envname.consul,cdh-master-2.node.envname.consul",
    "hadoop.proxyuser.httpfs.hosts": "*",
    "hadoop.security.authentication": "simple",
    "hbase.snapshot.master.timeout.millis": "60000",
    "hadoop.proxyuser.hdfs.hosts": "*",
    "hbase.client.write.buffer": "2097152",
    "dfs.client.use.legacy.blockreader": "false",
    "hadoop.proxyuser.hue.hosts": "*",
```

```
    "fs.permissions.umask-mode": "022",
    "hadoop.proxyuser.oozie.groups": "*",
    "dfs.ha.automatic-failover.enabled.nameservice1": "true",
    "hadoop.http.staticuser.user": "mapred",
    "hbase.zookeeper.property.clientPort": "2181",
    "hadoop.proxyuser.yarn.hosts": "*",
    "hadoop.proxyuser.flume.groups": "*",
    "hadoop.proxyuser.HTTP.groups": "*"
  },
```

4. 在 Java 代码里使用 hbase.zookeeper.quorum 及 hbase.zookeeper.property.clientPort 即可连接到 Hbase 数据库。

```
public class ConnectionPoolTest {
    private static final String QUORUM = "FBI001,FBI002,FBI003";
    private static final String CLIENTPORT = "2181";
    private static final String TABLENAME = "rd_ns:itable";
    private static Configuration conf = null;
    private static HConnection conn = null;

    static{
        try {
            conf =  HBaseConfiguration.create();
            conf.set("hbase.zookeeper.quorum", QUORUM);
            conf.set("hbase.zookeeper.property.clientPort", CLIENTPORT);
            conn = HConnectionManager.createConnection(conf);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Hdfs 概述

Hadoop 分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS 是一个高度容错性的系统，适合部署在廉价的机器上。HDFS 能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS 放宽了一部分 POSIX 约束，来实现流式读取文件系统数据的目的。HDFS 在最开始是作为 Apache Nutch 搜索引擎项目的基础架构而开发的。HDFS 是 Apache Hadoop Core 项目的一部分。

HDFS 有着高容错性(fault-tolerant)的特点，并且设计用来部署在低廉的(low-cost) 硬件上。而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。HDFS 放宽了（relax）POSIX 的要求（requirements）这样可以实现流的形式访问（streaming access）文件系统中的数据。

特点和目标

硬件故障

硬件故障是常态，而不是异常。整个 HDFS 系统将由数百或数千个存储着文件数据片断的服务器组成。实际上它里面有非常巨大的组成部分，每一个组成部分都很可能出现故障，这就意味着 HDFS 里的总是有一些部件是失效的，因此，故障的检测和自动快速恢复是 HDFS 一个很核心的设计目标。

数据访问

运行在 HDFS 之上的应用程序必须流式地访问它们的数据集，它不是运行在普通文件

系统之上的普通程序。HDFS 被设计成适合批量处理的，而不是用户交互式的。重点是在数据吞吐量，而不是数据访问的反应时间，POSIX 的很多硬性需求对于 HDFS 应用都是非必须的，去掉 POSIX 一小部分关键语义可以获得更好的数据吞吐率。

大数据集

运行在 HDFS 之上的程序有很大量的数据集。典型的 HDFS 文件大小是 GB 到 TB 的级别。所以，HDFS 被调整成支持大文件。它应该提供很高的聚合数据带宽，一个集群中支持数百个节点，一个集群中还应该支持千万级别的文件。

简单一致性模型

大部分的 HDFS 程序对文件操作需要的是一次写多次读取的操作模式。一个文件一旦创建、写入、关闭之后就不需要修改了。这个假定简单化了数据一致的问题，并使高吞吐量的数据访问变得可能。一个 Map-Reduce 程序或者网络爬虫程序都可以完美地适合这个模型。

移动计算比移动数据更经济

在靠近计算数据所存储的位置来进行计算是最理想的状态 ,尤其是在数据集特别巨大的时候。这样消除了网络的拥堵，提高了系统的整体吞吐量。一个假定就是迁移计算到离数据更近的位置比将数据移动到程序运行更近的位置要更好。HDFS 提供了接口，来让程序将自己移动到离数据存储更近的位置。

异构软硬件平台间的可移植性

HDFS 被设计成可以简便地实现平台间的迁移 ,这将推动需要大数据集的应用更广泛地采用 HDFS 作为平台。

名字节点和数据节点

HDFS 是一个主从结构，一个 HDFS 集群是由一个名字节点，它是一个管理文件命名空间和调节客户端访问文件的主服务器，当然还有一些数据节点，通常是一个节点一个机器，它来管理对应节点的存储。HDFS 对外开放文件命名空间并允许用户数据以文件形式存储。

内部机制是将一个文件分割成一个或多个块，这些块被存储在一组数据节点中。名字节点用来操作文件命名空间的文件或目录操作，如打开，关闭，重命名等等。它同时确定块与数据节点的映射。数据节点负责来自文件系统客户的读写请求。数据节点同时还要执行块的创建，删除，和来自名字节点的块复制指令。

名字节点和数据节点都是运行在普通的机器之上的软件，机器典型的都是 GNU/Linux，HDFS 是用 java 编写的，任何支持 java 的机器都可以运行名字节点或数据节点，利用 java 语言的超轻便型，很容易将 HDFS 部署到大范围的机器上。典型的部署是由一个专门的机器来运行名字节点软件，集群中的其他每台机器运行一个数据节点实例。体系结构不排斥在一个机器上运行多个数据节点的实例，但是实际的部署不会有这种情况。

集群中只有一个名字节点极大地简单化了系统的体系结构。名字节点是仲裁者和所有 HDFS 元数据的仓库，用户的实际数据不经过名字节点。

异常处理

可靠性

HDFS 的主要目标就是在存在故障的情况下也能可靠地存储数据。三个最常见的故障是名字节点故障，数据节点故障和网络断开。

重新复制

一个数据节点周期性发送一个心跳包到名字节点。网络断开会造成一组数据节点子集和名字节点失去联系。名字节点根据缺失的心跳信息判断故障情况。名字节点将这些数据节点标记为死亡状态，不再将新的 IO 请求转发到这些数据节点上，这些数据节点上的数据将对 HDFS 不再可用，可能会导致一些块的复制因子降低到指定的值。

名字节点检查所有的需要复制的块，并开始复制他们到其他的数据节点上。重新复制在有些情况下是不可或缺的，例如：数据节点失效，副本损坏，数据节点磁盘损坏或者文件的复制因子增大。

数据正确性

从数据节点上取一个文件块有可能是坏块，坏块的出现可能是存储设备错误，网络错误或者软件的漏洞。HDFS 客户端实现了 HDFS 文件内容的校验。当一个客户端创建一个 HDFS 文件时，它会为每一个文件块计算一个校验码并将校验码存储在同一个 HDFS 命名空间下一个单独的隐藏文件中。当客户端访问这个文件时，它根据对应的校验文件来验证从数据节点接收到的数据。如果校验失败，客户端可以选择从其他拥有该块副本的数据节点获取这个块。

元数据失效

FsImage 和 Editlog 是 HDFS 的核心数据结构。这些文件的损坏会导致整个集群的失效。因此，名字节点可以配置成支持多个 FsImage 和 EditLog 的副本。任何 FsImage 和 EditLog 的更新都会同步到每一份副本中。

同步更新多个 EditLog 副本会降低名字节点的命名空间事务交易速率。但是这种降低是可以接受的，因为 HDFS 程序中产生大量的数据请求，而不是元数据请求。名字节点重

新启动时，选择最新一致的 FsImage 和 EditLog。

名字节点对于一个 HDFS 集群是单点失效的。假如名字节点失效，就需要人工的干预。还不支持自动重启和到其它名字节点的切换。

特点

快照

快照支持在一个特定时间存储一个数据拷贝，快照可以将失效的集群回滚到之前一个正常的时间点上。HDFS 已经支持元数据快照。

数据组织

数据块

HDFS 的设计是用于支持大文件的。运行在 HDFS 上的程序也是用于处理大数据集的。这些程序仅写一次数据，一次或多次读数据请求，并且这些读操作要求满足流式传输速度。HDFS 支持文件的一次写多次读操作。HDFS 中典型的块大小是 64MB，一个 HDFS 文件可以被被切分成多个 64MB 大小的块，如果需要，每一个块可以分布在不同的数据节点上。

阶段状态

一个客户端创建一个文件的请求并不会立即转发到名字节点。实际上，一开始 HDFS 客户端将文件数据缓存在本地的临时文件中。应用程序的写操作被透明地重定向到这个临时本地文件。当本地文件堆积到一个 HDFS 块大小的时候，客户端才会通知名字节点。名字节点将文件名插入到文件系统层次中，然后为它分配一个数据块。名字节点构造包括数据节点 ID（可能是多个，副本数据块存放的节点也有）和目标数据块标识的报文，用它回复客户端的请求。客户端收到后将本地的临时文件刷新到指定的数据节点数据块中。

当文件关闭时，本地临时文件中未上传的残留数据就会被转送到数据节点。然后客户端就可以通知名字节点文件已经关闭。此时，名字节点将文件的创建操作添加到到持久化存储中。假如名字节点在文件关闭之前死掉，文件就丢掉了。

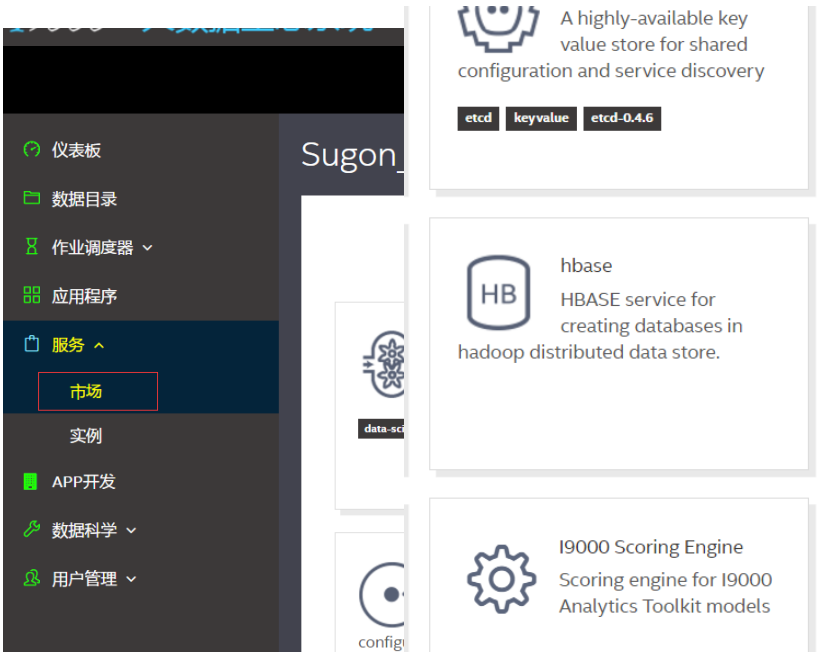
上述流程是在认真考虑了运行在 HDFS 上的目标程序之后被采用。这些应用程序需要流式地写文件。如果客户端对远程文件系统进行直接写入而没有任何本地的缓存，这就会对网速和网络吞吐量产生很大的影响。这方面早有前车之鉴，早期的分布式文件系统如 AFS，也用客户端缓冲来提高性能，POSIX 接口的限制也被放宽以达到更高的数据上传速率。

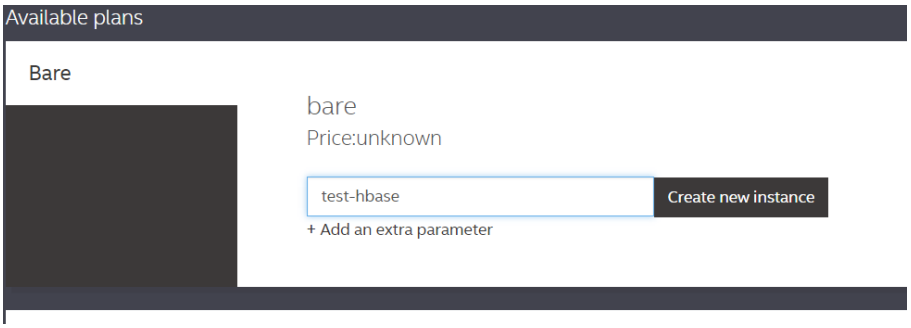
流水式复制

当客户端写数据到 HDFS 文件中时 ,如上所述 ,数据首先被写入本地文件中 ,假设 HDFS 文件的复制因子是 3，当本地文件堆积到一块大小的数据，客户端从名字节点获得一个数据节点的列表。这个列表也包含存放数据块副本的数据节点。当客户端刷新数据块到第一个数据节点。第一个数据节点开始以 4kb 为单元接收数据，将每一小块都写到本地库中，同时将每一小块都传送到列表中的第二个数据节点。同理，第二个数据节点将小块数据写入本地库中同时传给第三个数据节点，第三个数据节点直接写到本地库中。一个数据节点在接前一个节点数据的同时，还可以将数据流水式传递给下一个节点，所以，数据是流水式地从一个数据节点传递到下一个。

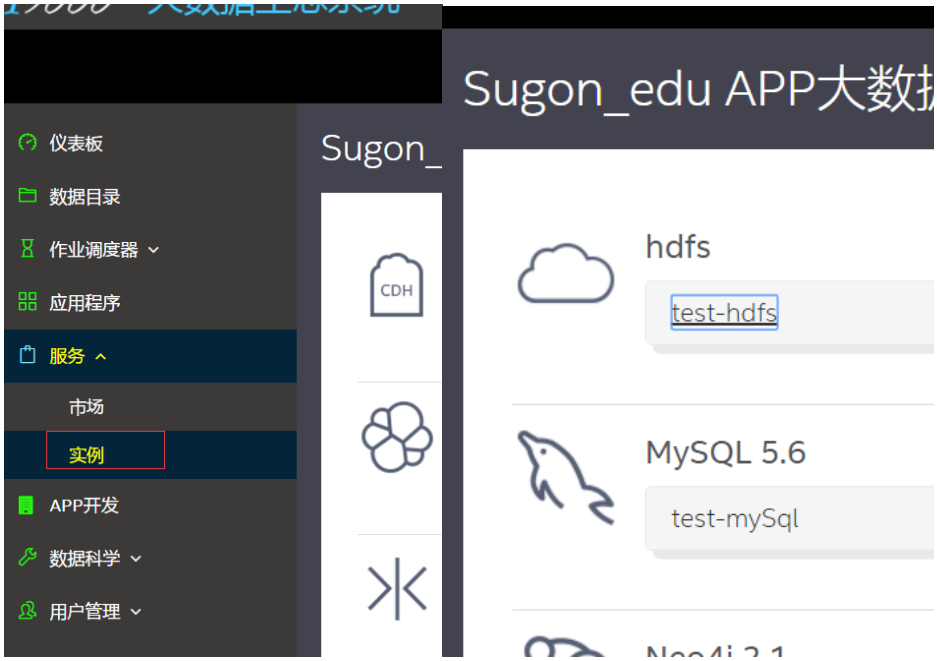
使用方法：

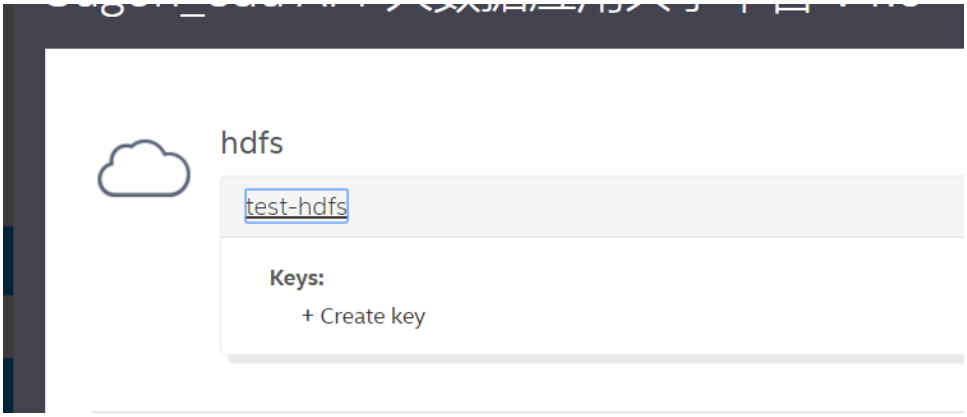
1. 在 i9000 平台上选择 服务>市场，找到 Hdfs，点击 Hdfs，创建一个新的实例。



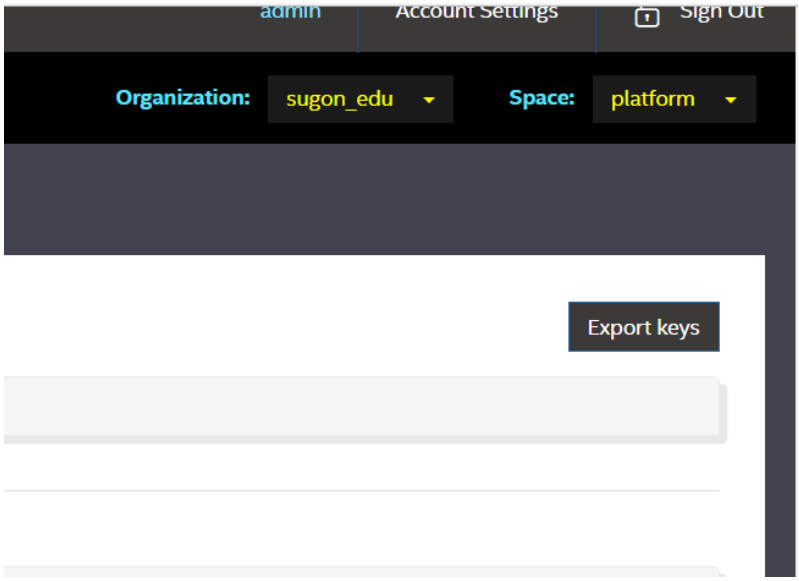


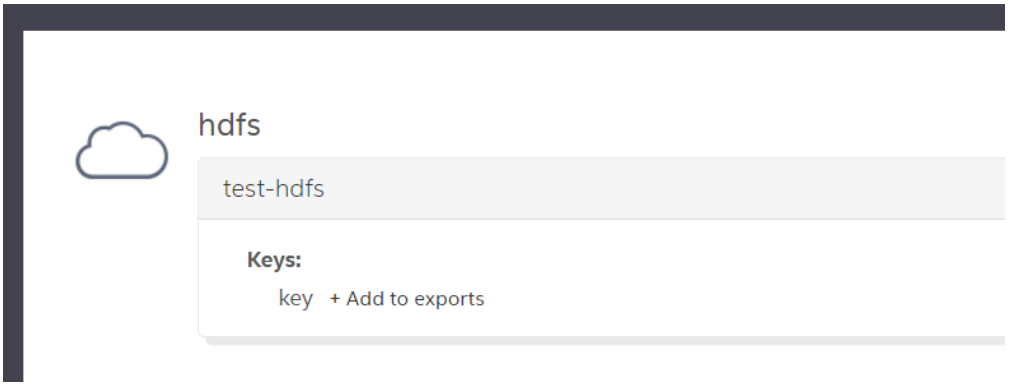
2. 在 服务>实例 里面找到 Hdfs，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Hdfs 的连接信息。





4. 在 Java 代码里使用 Hdfs 的 Java api 即可连接到 Hdfs 文件系统。

Hive 概述

hive 是基于 Hadoop 的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的 sql 查询功能，可以将 sql 语句转换为 MapReduce 任务进行运行。 其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析。

Hive 是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。

Hive 没有专门的数据格式。 Hive 可以很好的工作在 Thrift 之上，控制分隔符，也允许用户指定数据格式。

适用场景

Hive 构建在基于静态批处理的 Hadoop 之上，Hadoop 通常都有较高的延迟并且在作业提交和调度的时候需要大量的开销。因此，Hive 并不能够在大规模数据集上实现低延迟快速的查询，例如，Hive 在几百 MB 的数据集上执行查询一般有分钟级的时间延迟。因此，Hive 并不适合那些需要低延迟的应用，例如，联机事务处理（OLTP）。Hive 查询操作过程严格遵守 Hadoop MapReduce 的作业执行模型，Hive 将用户的 HiveQL 语句通过解释器转换为 MapReduce 作业提交到 Hadoop 集群上，Hadoop 监控作业执行过程，然后返回作业执行结果给用户。Hive 并非为联机事务处理而设计，Hive 并不提供实时的查询和基于行级的数据更新操作。Hive 的最佳使用场合是大数据集的批处理作业，例如，

网络日志分析。

设计特征

Hive 是一种底层封装了 Hadoop 的数据仓库处理工具，使用类 SQL 的 HiveQL 语言实现数据查询，所有 Hive 的数据都存储在 Hadoop 兼容的文件系统（例如，Amazon S3、HDFS）中。Hive 在加载数据过程中不会对数据进行任何的修改，只是将数据移动到 HDFS 中 Hive 设定的目录下，因此，Hive 不支持对数据的改写和添加，所有的数据都是在加载的时候确定的。Hive 的设计特点如下。

- 支持索引，加快数据查询。
- 不同的存储类型，例如，纯文本文件、HBase 中的文件。
- 将元数据保存在关系数据库中，大大减少了在查询过程中执行语义检查的时间。
- 可以直接使用存储在 Hadoop 文件系统中的数据。
- 内置大量用户函数 UDF 来操作时间、字符串和其他的数据挖掘工具，支持用户扩展 UDF 函数来完成内置函数无法实现的操作。
- 类 SQL 的查询方式 将 SQL 查询转换为 MapReduce 的 job 在 Hadoop 集群上执行。

Hive 体系结构

主要分为以下几个部分：

用户接口

用户接口主要有三个：CLI，Client 和 WUI。其中最常用的是 CLI，Cli 启动的时候，会同时启动一个 Hive 副本。Client 是 Hive 的客户端，用户连接至 Hive Server。在启

动 Client 模式的时候，需要指出 Hive Server 所在节点，并且在该节点启动 Hive Server。 WUI 是通过浏览器访问 Hive。

元数据存储

Hive 将元数据存储在数据库中，如 mysql、derby。Hive 中的元数据包括表的名字，表的列和分区及其属性，表的属性（是否为外部表等），表的数据所在目录等。

解释器、编译器、优化器、执行器

解释器、编译器、优化器完成 HQL 查询语句从词法分析、语法分析、编译、优化以及查询计划的生成。生成的查询计划存储在 HDFS 中，并在随后由 MapReduce 调用执行。

Hive 的数据存储在 HDFS 中，大部分的查询由 MapReduce 完成（包含 * 的查询，比如 select * from tbl 不会生成 MapReduce 任务）。

数据存储

首先，Hive 没有专门的数据存储格式，也没有为数据建立索引，用户可以非常自由的组织 Hive 中的表，只需要在创建表的时候告诉 Hive 数据中的列分隔符和行分隔符，Hive 就可以解析数据。

其次，Hive 中所有的数据都存储在 HDFS 中，Hive 中包含以下数据模型：表(Table)，外部表(External Table)，分区(Partition)，桶(Bucket)。

Hive 中的 Table 和数据库中的 Table 在概念上是类似的，每一个 Table 在 Hive 中都有一个相应的目录存储数据。例如，一个表 pvs，它在 HDFS 中的路径为：/wh/pvs，其中，wh 是在 hive-site.xml 中由 ${hive.metastore.warehouse.dir} 指定的数据仓库

的目录，所有的 Table 数据（不包括 External Table）都保存在这个目录中。

Partition 对应于数据库中的 Partition 列的密集索引，但是 Hive 中 Partition 的组织方式和数据库中的很不相同。在 Hive 中，表中的一个 Partition 对应于表下的一个目录，所有的 Partition 的数据都存储在对应的目录中。例如：pvs 表中包含 ds 和 city 两个 Partition，则对应于 ds = 20090801, ctry = US 的 HDFS 子目录为：/wh/pvs/ds=20090801/ctry=US；

对应于 ds = 20090801, ctry = CA 的 HDFS 子目录为：

/wh/pvs/ds=20090801/ctry=CA

Buckets 对指定列计算 hash，根据 hash 值切分数据，目的是为了并行，每一个 Bucket 对应一个文件。将 user 列分散至 32 个 bucket，首先对 user 列的值计算 hash，对应 hash 值为 0 的 HDFS 目录为：/wh/pvs/ds=20090801/ctry=US/part-00000；hash 值为 20 的 HDFS 目录为：/wh/pvs/ds=20090801/ctry=US/part-00020

External Table 指向已经在 HDFS 中存在的数据，可以创建 Partition。它和 Table 在元数据的组织上是相同的，而实际数据的存储则有较大的差异。

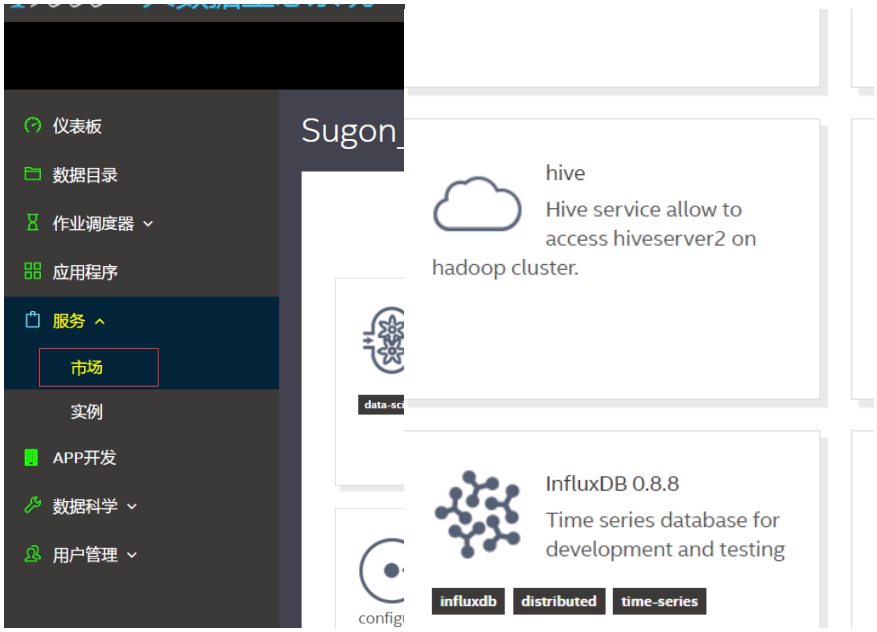
Table 的创建过程和数据加载过程（这两个过程可以在同一个语句中完成），在加载数据的过程中，实际数据会被移动到数据仓库目录中；之后对数据对访问将会直接在数据仓库目录中完成。删除表时，表中的数据和元数据将会被同时删除。

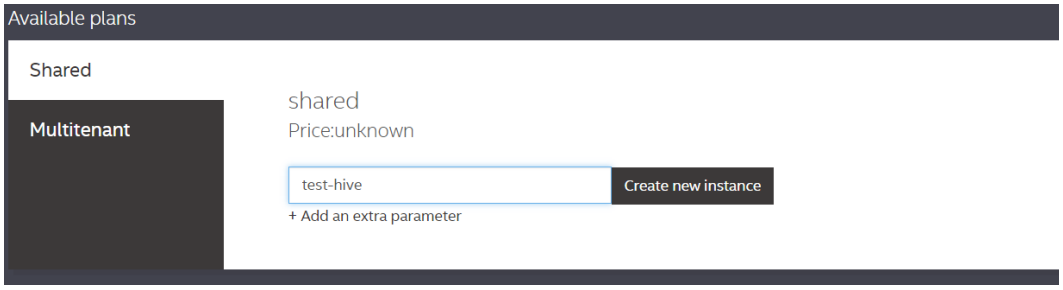
External Table 只有一个过程，加载数据和创建表同时完成（CREATE EXTERNAL TABLE ……LOCATION），实际数据是存储在 LOCATION 后面指定的 HDFS 路径中，并

不会移动到数据仓库目录中。当删除一个 External Table 时，仅删除元数据，表中的数据不会真正被删除。

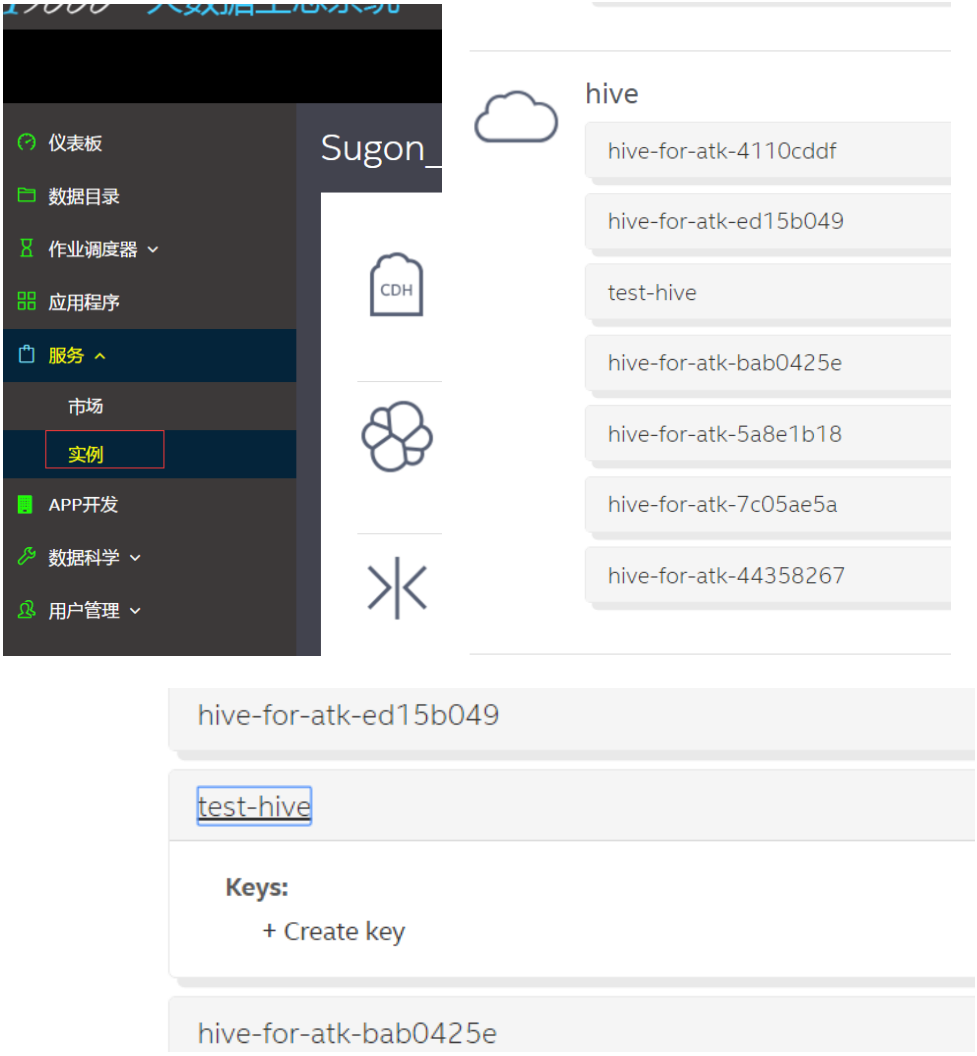
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 Hive，点击 Hive，创建一个新的实例。

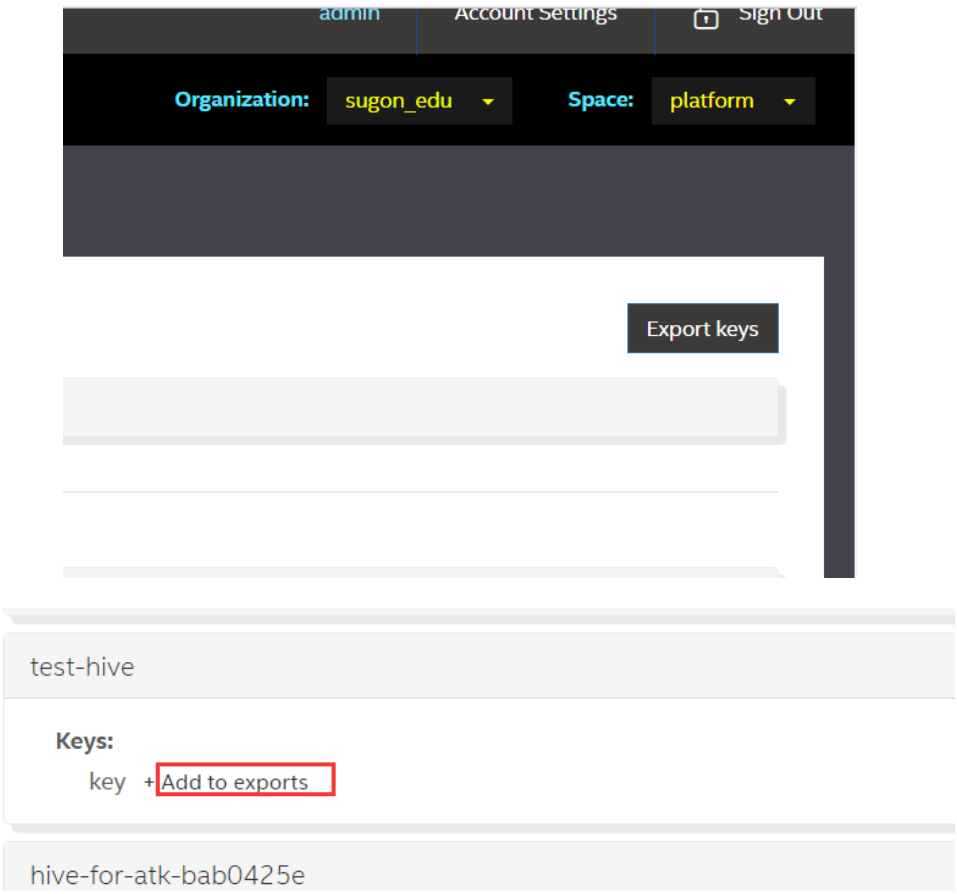




2. 在 服务>实例 里面找到 Hive，找到刚刚创建的实例，点击，创建新的 key。



3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Hive 的配置信息。



a) 在平台上 Hive 可以与其他应用绑定使用，通过读取环境变量的方式获取 Hive 的配置信息。

InfluxDB 0.8.8 概述

Influxdb 是一个开源的分布式时序、时间和指标数据库，使用 go 语言编写，无需外部依赖。

它有三大特性：

时序性（Time Series）：与时间相关的函数的灵活使用（诸如最大、最小、求和等）；

度量（Metrics）：对实时大量数据进行计算；

事件（Event）：支持任意的事件数据，换句话说，任意事件的数据我们都可以做操作。

同时，它有以下几大特点：

schemaless(无结构)，可以是任意数量的列；

min, max, sum, count, mean, median 一系列函数，方便统计；

Native HTTP API, 内置 http 支持，使用 http 读写；

Powerful Query Language 类似 sql；

Built-in Explorer 自带管理工具。

技术概述

InfluxDB 提供了一种类似 SQL 的语言，内置时间中心函数，用于查询由测量，序列和点组成的数据结构。CEach 点由几个称为字段集和时间戳的键值对组成。 当通过一组称为标签集的键值对组合在一起时，这些定义一个系列。 最后，通过字符串标识符将系列分组在一起以形成测量。

值可以是 64 位整数，64 位浮点数，字符串和布尔值。

点按其时间和标签集编制索引。

保留策略是对测量定义的，并且控制如何对数据进行降采样和删除。

连续查询定期运行，将结果存储在目标测量中。

线路协议

InfluxDB 通过 HTTP，TCP 和 UDP 接受数据。

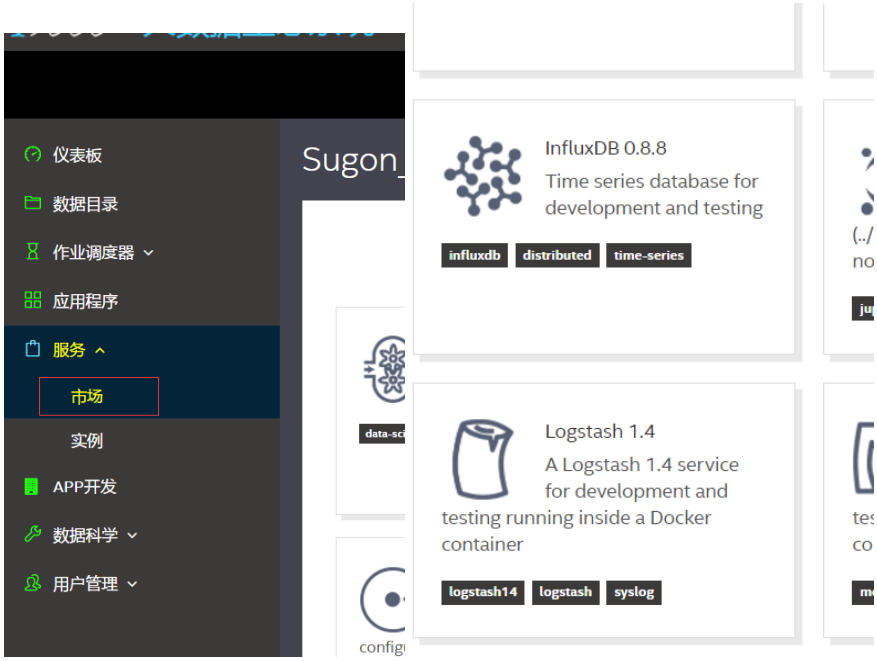
它定义了与石墨向后兼容的线路协议，形式如下：

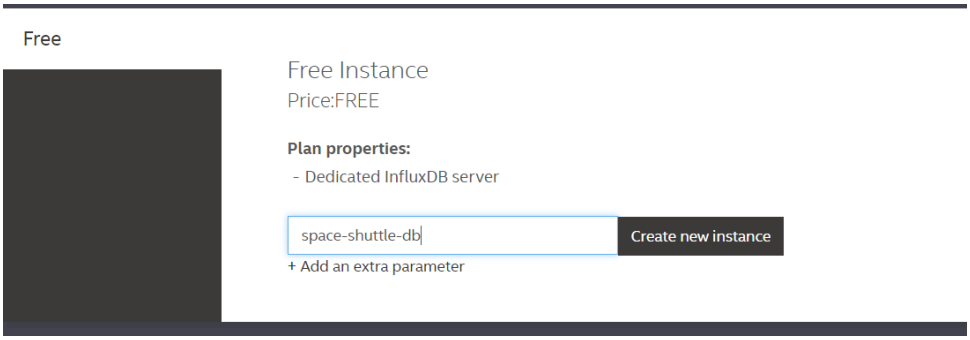
measurement(,tag_key=tag_val)*　　field_key=field_val(,field_key_n=field_value_n)*

(nanoseconds-timestamp)?

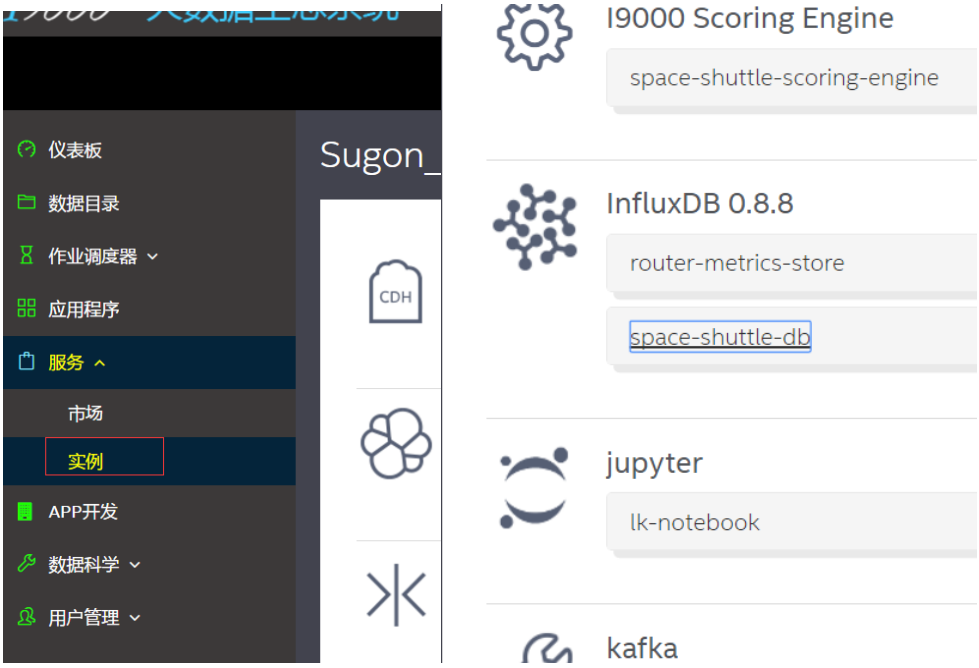
使用方法：

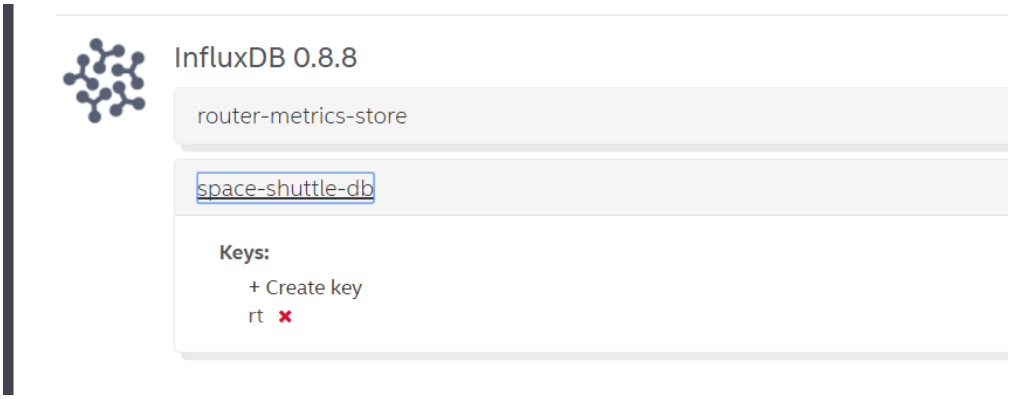
1. 在 i9000 平台上选择 服务>市场，找到 InfluxDB，点击 InfluxDB，创建一个新的实例。



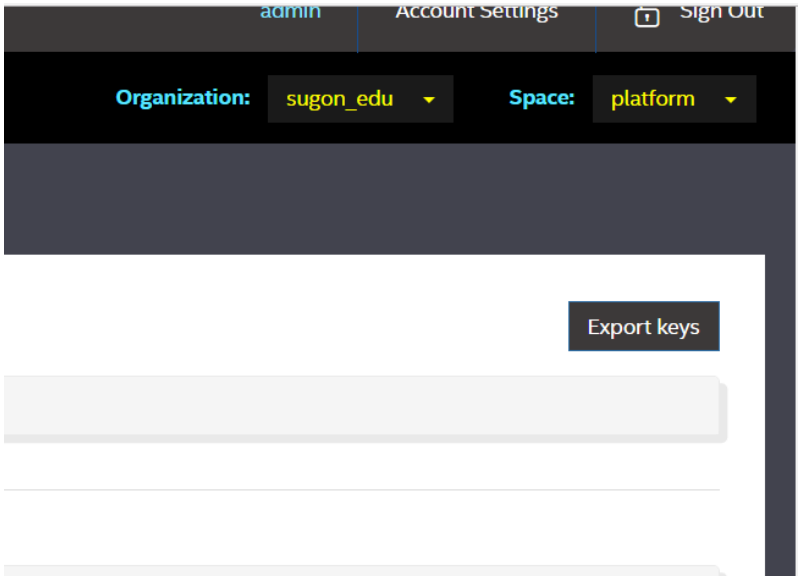


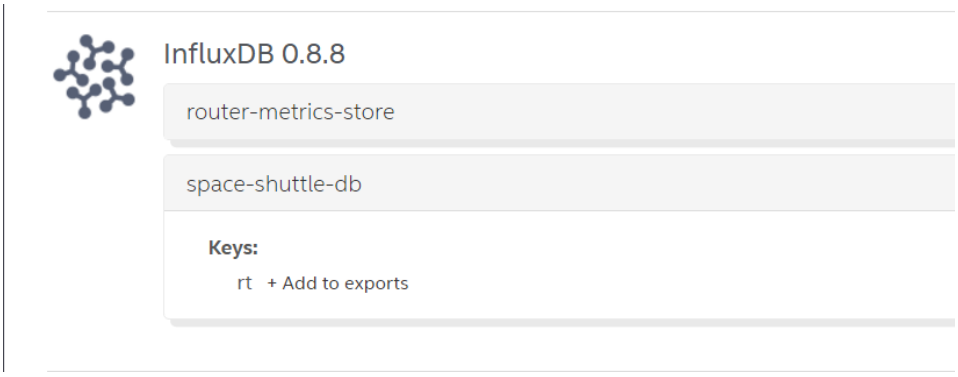
2. 在 服务>实例 里面找到 InfluxDB，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 InfluxDB 的连接信息。



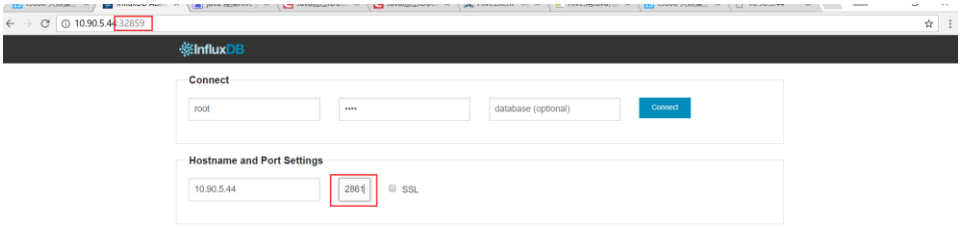


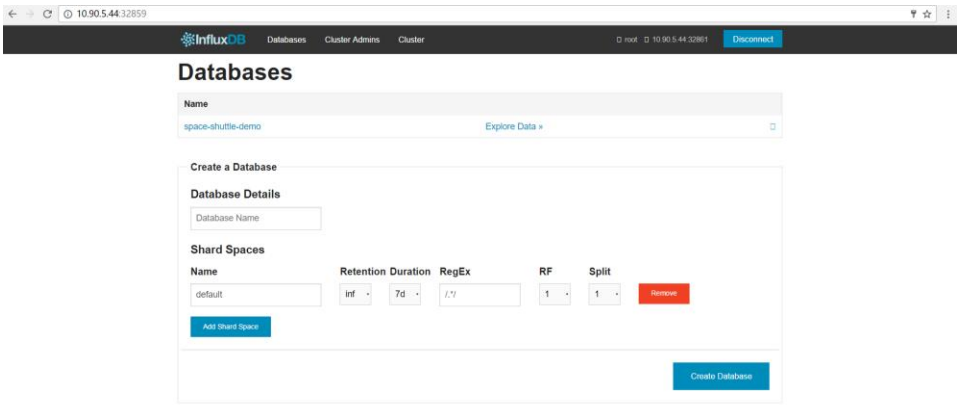
exported Keys:


```
{
  "influxdb088": [
    {
      "label": "influxdb088",
      "name": "space-shuttle-db",
      "plan": "free",
      "tags": [
        "influxdb",
        "distributed",
        "time-series"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "8083/tcp": "32859",
          "8084/tcp": "32860",
          "8086/tcp": "32861"
        },
        "port": "32861",
        "username": "root",
        "password": "root",
        "uri": "http://root:root@10.0.4.4:32861"
      }
    }
  ]
}
```


4. 在浏览器里输入 hostname:port 进行访问，可以看到界面化的 InfluxDB,输入

username 和 password 可以登录（这里的 Key 提供了 3 个 port，分别是 8083，8084，8086 映射的端口，浏览器地址栏里输入 8083 映射的端口，登录界面输入 8086 映射的端口）。





RabbitMQ 3.3 概述

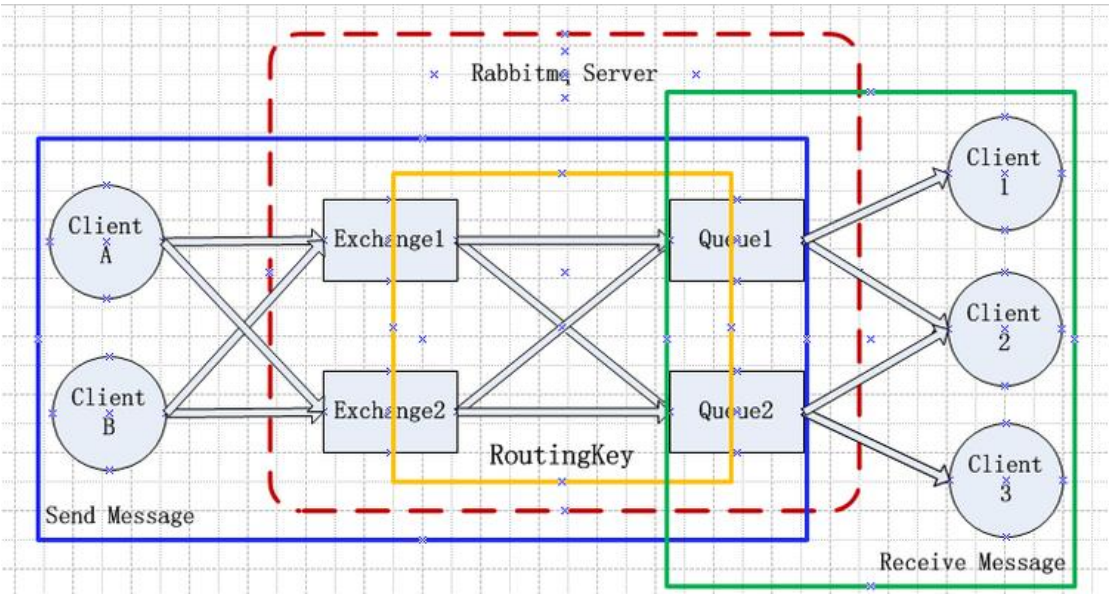
MQ 全称为 Message Queue，消息队列（MQ）是一种应用程序对应用程序的通信方法。应用程序通过读写出入队列的消息（针对应用程序的数据）来通信，而无需专用连接来链接它们。消息传递指的是程序之间通过在消息中发送数据进行通信，而不是通过直接调用彼此来通信，直接调用通常是用于诸如远程过程调用的技术。排队指的是应用程序通过 队列来通信。队列的使用除去了接收和发送应用程序同时执行的要求。其中较为成熟的 MQ 产品有 IBM WEBSPHERE MQ 等等。

MQ 特点

MQ 是消费-生产者模型的一个典型的代表，一端往消息队列中不断写入消息，而另一端则可以读取或者订阅队列中的消息。MQ 和 JMS 类似，但不同的是 JMS 是 SUN JAVA 消息中间件服务的一个标准和 API 定义，而 MQ 则是遵循了 AMQP 协议的具体实现和产品。

RabbitMQ 基本概念

RabbitMQ 是流行的开源消息队列系统，用 erlang 语言开发。RabbitMQ 是 AMQP（高级消息队列协议）的标准实现。如果不熟悉 AMQP，直接看 RabbitMQ 的文档会比较困难。不过它也只有几个关键概念，这里简单介绍。



几个概念说明：

Broker：简单来说就是消息队列服务器实体。

Exchange：消息交换机，它指定消息按什么规则，路由到哪个队列。

Queue：消息队列载体，每个消息都会被投入到一个或多个队列。

Binding：绑定，它的作用就是把 exchange 和 queue 按照路由规则绑定起来。

Routing Key：路由关键字，exchange 根据这个关键字进行消息投递。

vhost：虚拟主机，一个 broker 里可以开设多个 vhost，用作不同用户的权限分离。

producer：消息生产者，就是投递消息的程序。

consumer：消息消费者，就是接受消息的程序。

channel：消息通道，在客户端的每个连接里，可建立多个 channel，每个 channel 代表一个会话任务。

消息队列的使用过程大概如下：

（1）客户端连接到消息队列服务器，打开一个 channel。

（2）客户端声明一个 exchange，并设置相关属性。

（3）客户端声明一个 queue，并设置相关属性。

（4）客户端使用 routing key，在 exchange 和 queue 之间建立好绑定关系。

（5）客户端投递消息到 exchange。

exchange 接收到消息后，就根据消息的 key 和已经设置的 binding，进行消息路由，将消息投递到一个或多个队列里。

exchange 也有几个类型，完全根据 key 进行投递的叫做 Direct 交换机，例如，绑定时设置了 routing key 为”abc”，那么客户端提交的消息，只有设置了 key 为”abc”的才会投递到队列。对 key 进行模式匹配后进行投递的叫做 Topic 交换机，符号”#”匹配一个或多个词，符号”*”匹配正好一个词。例如”abc.#”匹配”abc.def.ghi”，”abc.*”只匹配”abc.def”。还有一种不需要 key 的，叫做 Fanout 交换机，它采取广播模式，一个消息进来时，投递到与该交换机绑定的所有队列。

RabbitMQ 支持消息的持久化，也就是数据写在磁盘上，为了数据安全考虑，我想大多数用户都会选择持久化。消息队列持久化包括 3 个部分：

（1）exchange 持久化，在声明时指定 durable => 1

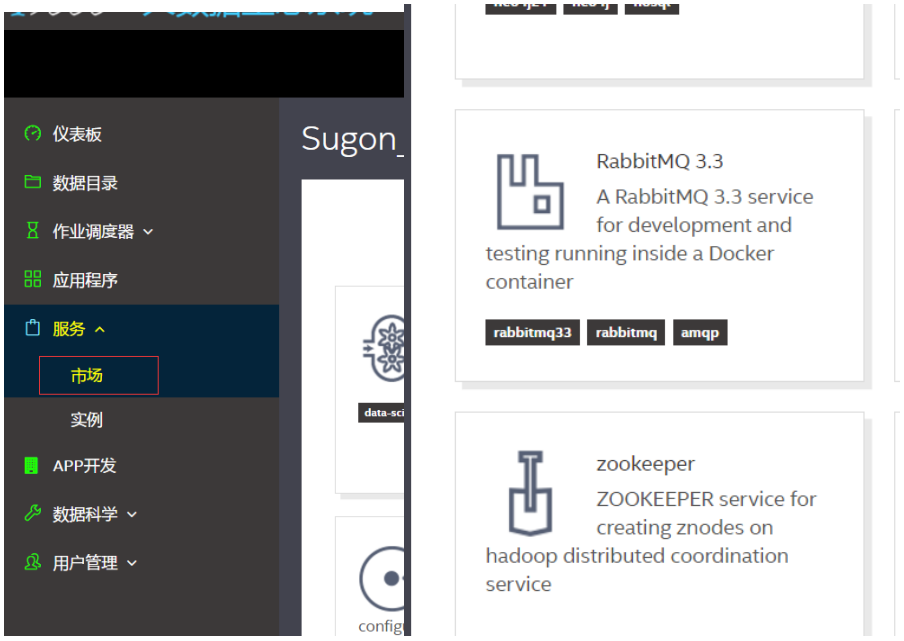
（2）queue 持久化，在声明时指定 durable => 1

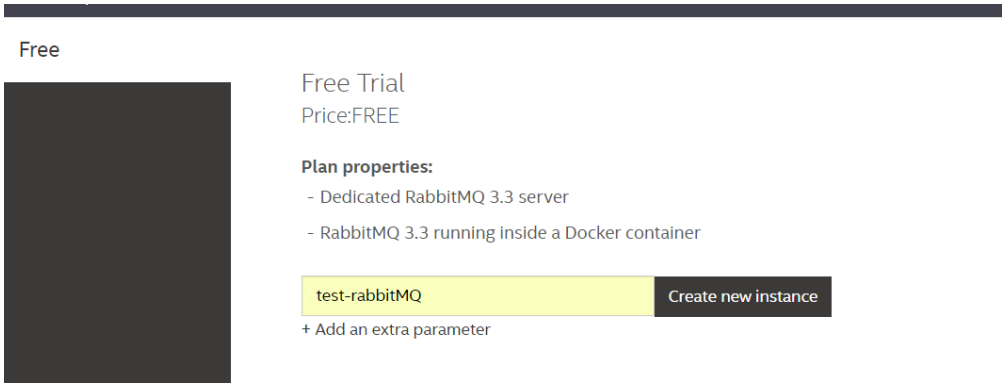
（3）消息持久化，在投递时指定 delivery_mode => 2（1 是非持久化）

如果 exchange 和 queue 都是持久化的，那么它们之间的 binding 也是持久化的。如果 exchange 和 queue 两者之间有一个持久化，一个非持久化，就不允许建立绑定。

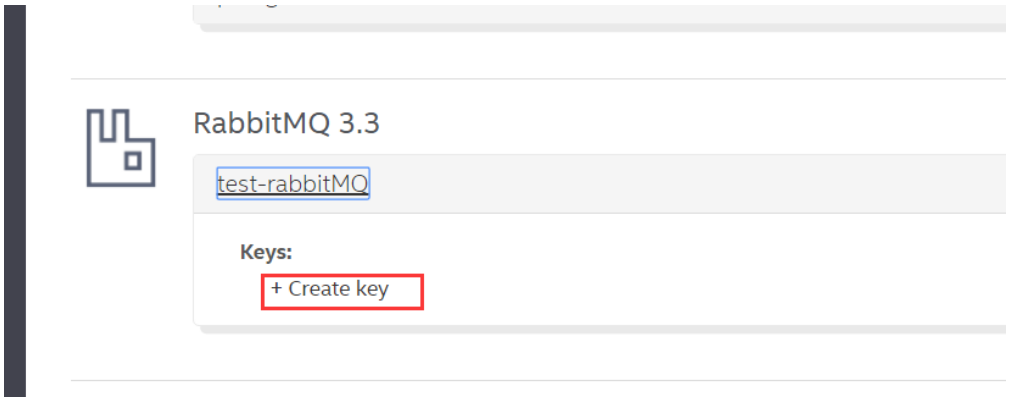
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 RabbitMQ，点击 RabbitMQ，创建一个新的实例。

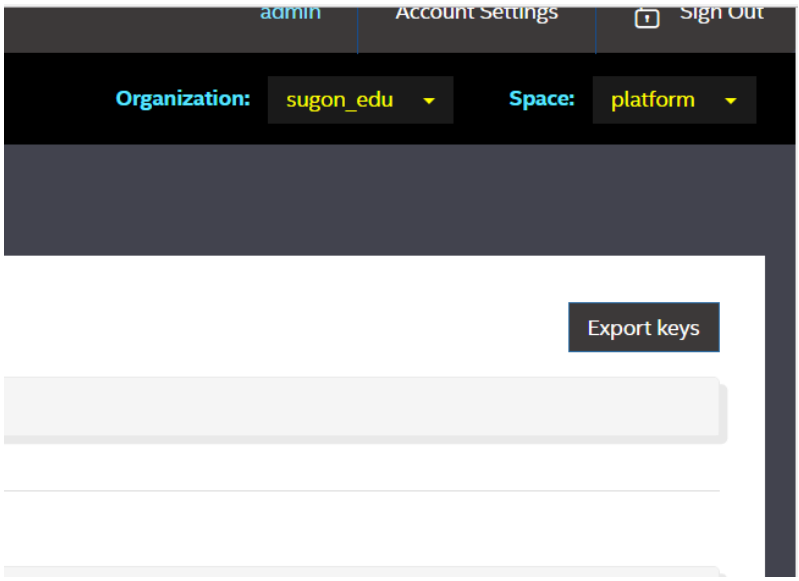


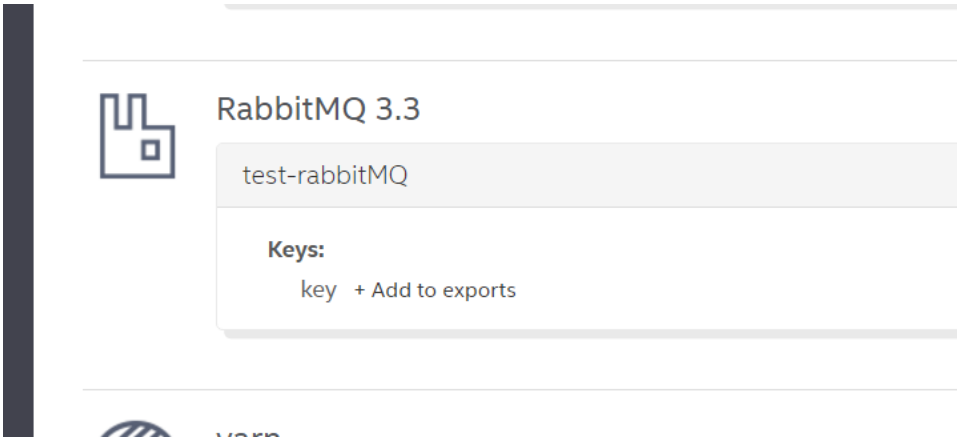


2. 在 服务>实例 里面找到 RabbitMQ，找到刚刚创建的实例，点击，创建新的 key。



3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 RabbitMQ 的连接信息。

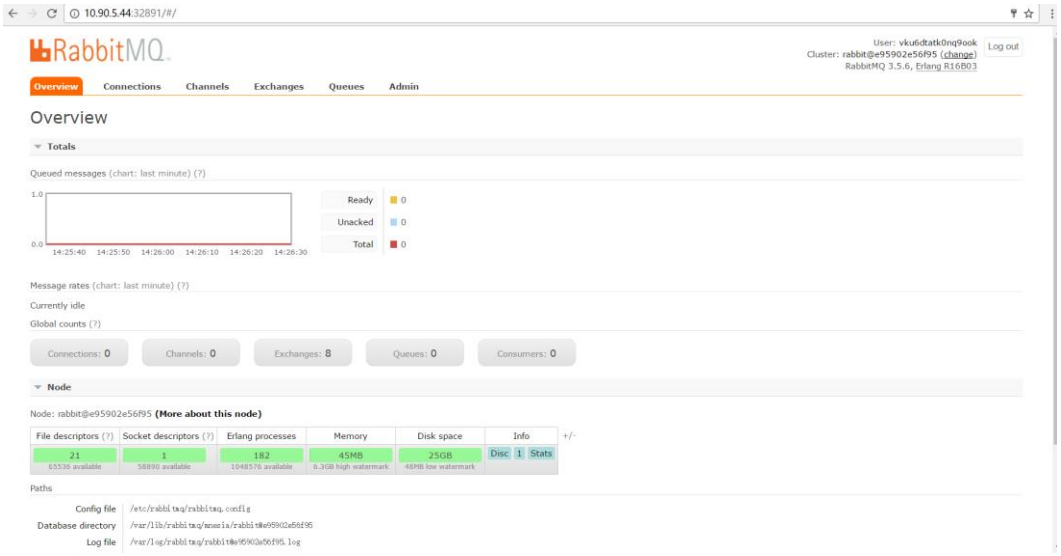




exported Keys:


```
{
  "rabbitmq33": [
    {
      "label": "rabbitmq33",
      "name": "test-rabbitMQ",
      "plan": "free",
      "tags": [
        "rabbitmq33",
        "rabbitmq",
        "amqp"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "15672/tcp": "32891",
          "5672/tcp": "32892"
        },
        "port": "32892",
        "username": "vku6dtatk0nq9ook",
        "password": "gwu960hd23f9kcv2",
        "uri": "amqp://vku6dtatk0nq9ook:gwu960hd23f9kcv2@10.0.4.4:32892"
      }
    }
  ]
}
```


4. 在浏览器里输入 hostname:port（15672 映射的端口）进行访问，可以看到界面化的 RabbitMQ,输入 username 和 password 可以登录。


10.90.5.44:32891/#/
RabbitMQ
User: vku6dtatk0nq9ook
Cluster: rabbit@e95902e56f95 (change)
RabbitMQ 3.5.6, Erlang R16B03
Log out
Overview
Connections
Channels
Exchanges
Queues
Admin
Overview
Totals
Queued messages (chart: last minute) (?)
1.0
0.0
14:25:40 14:25:50 14:26:00 14:26:10 14:26:20 14:26:30
Ready 0
Unacked 0
Total 0
Message rates (chart: last minute) (?)
Currently idle
Global counts (?)
Connections: 0
Channels: 0
Exchanges: 8
Queues: 0
Consumers: 0
Node
Node: rabbit@e95902e56f95 (More about this node)
File descriptors (?)
Socket descriptors (?)
Erlang processes
Memory
Disk space
Info
21
65536 available
1
58890 available
182
1048576 available
45MB
6.3GB high watermark
25GB
48MB low watermark
Disc 1 Stats
Paths
Config file /etc/rabbitmq/rabbitmq.config
Database directory /var/lib/rabbitmq/mnesia/rabbit@e95902e56f95
Log file /var/log/rabbitmq/rabbit@e95902e56f95.log

Redis2.8 概述

redis 是一个 key-value 存储系统。和 Memcached 类似，它支持存储的 value 类型相对更多，包括 string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和 hash（哈希类型）。这些数据类型都支持 push/pop、add/remove 及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis 支持各种不同方式的排序。与 memcached 一样，为了保证效率，数据都是缓存在内存中。区别的是 redis 会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了 master-slave(主从)同步。

Redis 是一个高性能的 key-value 数据库。 redis 的出现，很大程度补偿了 memcached 这类 key/value 存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了 Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang 等客户端，使用很方便。

Redis 支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得 Redis 可执行单层树复制。存盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

数据模型

Redis 的外围由一个键、值映射的字典构成。与其他非关系型数据库主要不同在于：Redis 中值的类型[1] 不仅限于字符串，还支持如下抽象数据类型：

字符串列表

无序不重复的字符串集合

有序不重复的字符串集合

键、值都为字符串的哈希表

值的类型决定了值本身支持的操作。Redis 支持不同无序、有序的列表，无序、有序的集合间的交集、并集等高级服务器端原子操作。

存储

redis 使用了两种文件格式：全量数据和增量请求。全量数据格式是把内存中的数据写入磁盘 ,便于下次读取文件进行加载 ,增量请求文件则是把内存中的数据序列化为操作请求，用于读取文件进行 replay 得到数据，序列化的操作包括 SET、RPUSH、SADD、ZADD。

redis 的存储分为内存存储、磁盘存储和 log 文件三部分，配置文件中有三个参数对其进行配置。

save seconds updates，save 配置，指出在多长时间内，有多少次更新操作，就将数据同步到数据文件。这个可以多个条件配合，比如默认配置文件中的设置，就设置了三个条件。

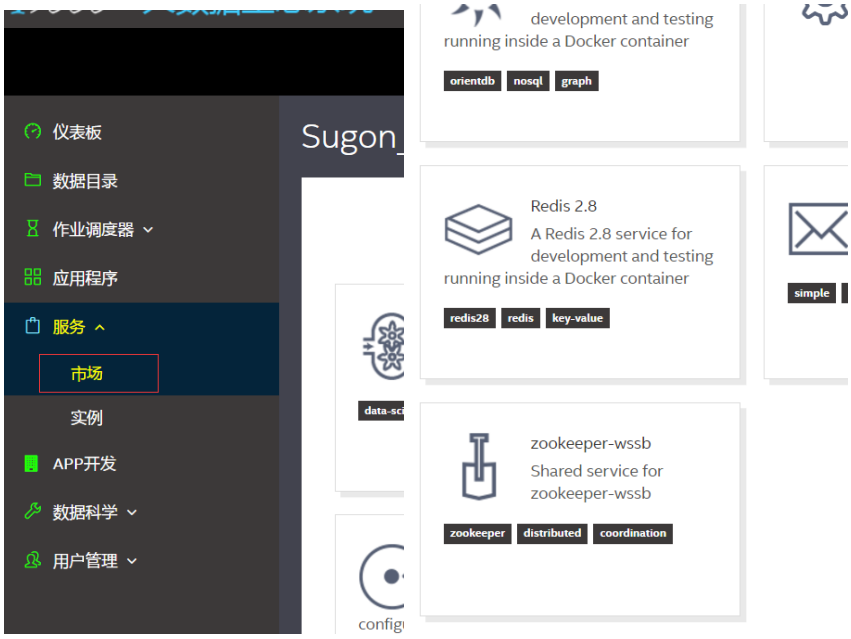
appendonly yes/no ,appendonly 配置 ,指出是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为 redis 本身同步数据文件是按上面的 save 条件来同步的，所以有的数据会在一段时间内只存在于内存中。

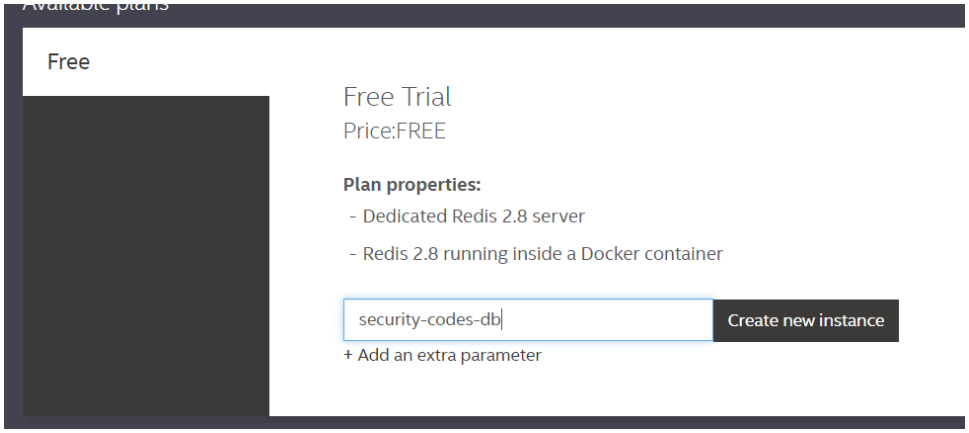
appendfsync no/always/everysec ，appendfsync 配置，no 表示等操作系统进行数据缓存同步到磁盘，always 表示每次更新操作后手动调用 fsync()将数据写到磁盘，

everysec 表示每秒同步一次。

使用方法：

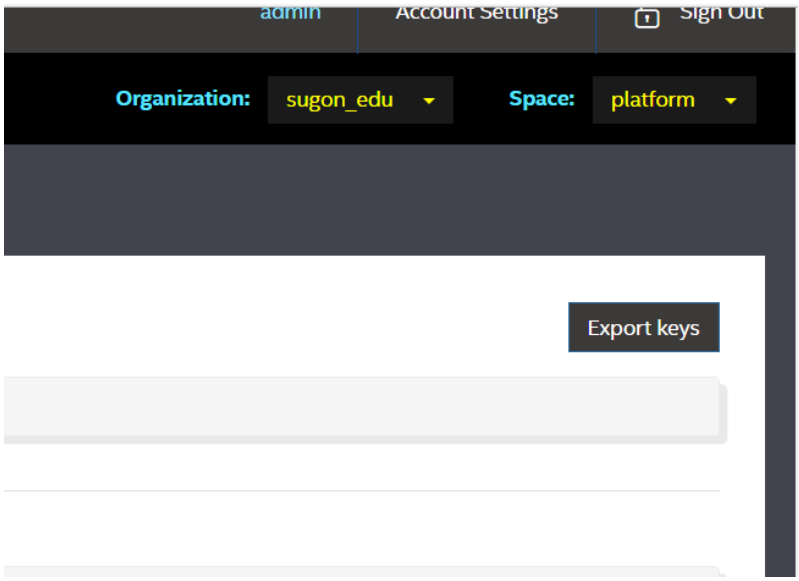
1. 在 i9000 平台上选择 服务>市场，找到 Redis2.8，点击 Redis2.8，创建一个新的实例。

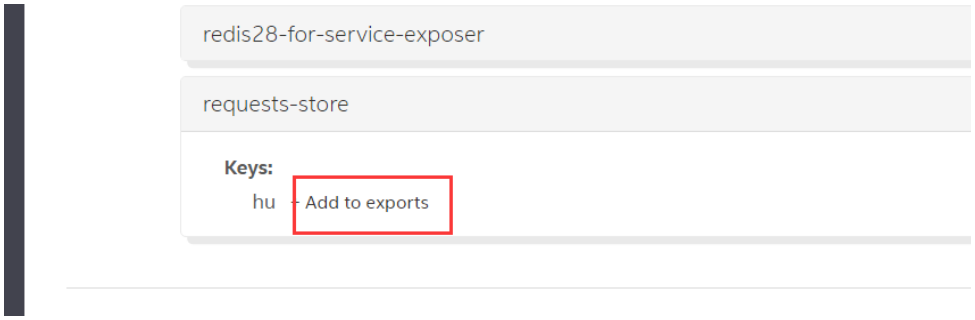




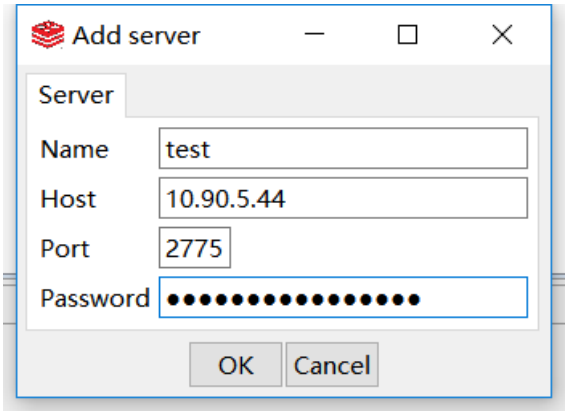
2. 在 服务>实例 里面找到 Redis2.8，找到刚刚创建的实例，点击，创建新的 key。

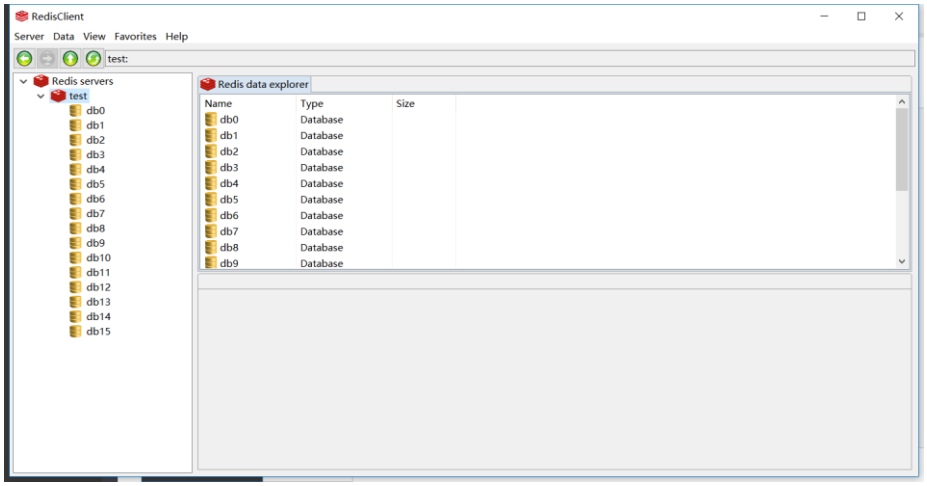
3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Redis2.8 的连接信息。





4. 在 RedisClient 客户端里，可以通过 hostname，port 以及 password 连接到刚刚创建的 Redis 数据库


Add server
Server
Name
test
Host
10.90.5.44
Port
2775
Password
OK
Cancel


RedisClient
Server Data View Favorites Help
test:
Redis servers
test
db0
db1
db2
db3
db4
db5
db6
db7
db8
db9
db10
db11
db12
db13
db14
db15
Redis data explorer
Name
Type
Size
Database

SMTP

概述

SMTP (Simple Mail Transfer Protocol) 即简单邮件传输协议,它是一组用于由源地址到目的地址传送邮件的规则，由它来控制信件的中转方式。SMTP 协议属于 TCP/IP 协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。通过 SMTP 协议所指定的服务器,就可以把 E-mail 寄到收信人的服务器上了，整个过程只要几分钟。SMTP 服务器则是遵循 SMTP 协议的发送邮件服务器，用来发送或中转发出的电子邮件。

它使用由 TCP 提供的可靠的数据传输服务把邮件消息从发信人的邮件服务器传送到收信人的邮件服务器。跟大多数应用层协议一样，SMTP 也存在两个端：在发信人的邮件服务器上执行的客户端和在收信人的邮件服务器上执行的服务器端。SMTP 的客户端和服务器端同时运行在每个邮件服务器上。当一个邮件服务器在向其他邮件服务器发送邮件消息时，它是作为 SMTP 客户在运行。

SMTP 协议与人们用于面对面交互的礼仪之间有许多相似之处。首先，运行在发送端邮件服务器主机上的 SMTP 客户，发起建立一个到运行在接收端邮件服务 器主机上的 SMTP 服务器端口号 25 之间的 TCP 连接。如果接收邮件服务器当前不在工作，SMTP 客户就等待一段时间后再尝试建立该连接。SMTP 客户和服务器先执行一些应用层握手操作。就像人们在转手东西之前往往先自我介绍那样 ,SMTP 客户和服务器也在传送信息之前先自我介绍一下。 在这个 SMTP 握手阶段，SMTP 客户向服务器分别指出发信人和收信人的电子邮件地址。彼此自我介绍完毕之后，客户发出邮件消息。

SMTP 是一种 TCP 协议支持的提供可靠且有效电子邮件传输的应用层协议。SMTP 是建立在 TCP 上的一种邮件服务 ,主要用于传输系统之间的邮件信息并提供来信有关的通知。

SMTP 独立于特定的传输子系统，且只需要可靠有序的数据流信道支持。SMTP 重要特性之一是其能跨越网络传输邮件，即“ SMTP 邮件中继”。通常，一个网络可以由公用互联网上 TCP 可相互访问的主机、防火墙分隔的 TCP/IP 网络上 TCP 可相互访问的主机，及其它 LAN/WAN 中的主机利用非 TCP 传输层协议组成。使用 SMTP ，可实现相同网络上处理机之间的邮件传输 ,也可通过中继器或网关实现某处理机与其它网络之间的邮件传输。在这种方式下，邮件的发送可能经过从发送端到接收端路径上的大量中间中继器或网关主机。域名服务系统（DNS）的邮件交换服务器可以用来识别出传输邮件的下一条 IP 地址。

SMTP 是一个相对简单的基于文本的协议。在其之上指定了一条消息的一个或多个接收者（在大多数情况下被确认是存在的），然后消息文本会被传输。可以很简单地通过 telnet 程序来测试一个 SMTP 服务器。SMTP 使用 TCP 端口 25。要为一个给定的域名决定一个 SMTP 服务器，需要使用 MX (Mail eXchange)DNS。

在八十年代早期 SMTP 开始被广泛地使用。当时，它只是作为 UUCP 的补充，UUCP 更适合于处理在间歇连接的机器间传送邮件。相反，SMTP 在发送和接收的机器始终连接在网络的情况下工作得最好。

Sendmail 是最早实现 SMTP 的邮件传输代理之一。到 2001 年至少有 50 个程序将 SMTP 实现为一个客户端（消息的发送者）或一个服务器（消息的接收者）。一些其他的流行的 SMTP 服务器程序包括了 Philip Hazel 的 exim，IBM 的 Postfix，D. J. Bernstein 的 Qmail，以及 Microsoft Exchange Server。

由于这个协议开始是基于纯 ASCII 文本的，它在二进制文件上处理得并不好。诸如 MIME 的标准被开发来编码二进制文件以使其通过 SMTP 来传输。今天，大多数 SMTP 服

务器都支持 8 位 MIME 扩展，它使二进制文件的传输变得几乎和纯文本一样简单。

SMTP 是一个“推”的协议，它不允许根据需要从远程服务器上“拉”来消息。要做到这点，邮件客户端必须使用 POP3 或 IMAP。另一个 SMTP 服务器可以使用 ETRN 在 SMTP 上触发一个发送。

工作过程

简单邮件传输协议（SMTP）是一种基于文本的电子邮件传输协议，是在因特网中用于在邮件服务器之间交换邮件的协议。SMTP 是应用层的服务，可以适应于各种网络系统。

SMTP 的命令和响应都是基于文本，以命令行为单位，换行符为 CR/LF。响应信息一般只有一行，由一个 3 位数的代码开始，后面可附上很简短的文字说明。

SMTP 要经过建立连接、传送邮件和释放连接 3 个阶段。具体为：

（1）建立 TCP 连接。

（2）客户端向服务器发送 HELO 命令以标识发件人自己的身份，然后客户端发送 MAIL 命令。

（3）服务器端以 OK 作为响应，表示准备接收。

（4）客户端发送 RCPT 命令。

（5）服务器端表示是否愿意为收件人接收邮件。

（6）协商结束，发送邮件，用命令 DATA 发送输入内容。

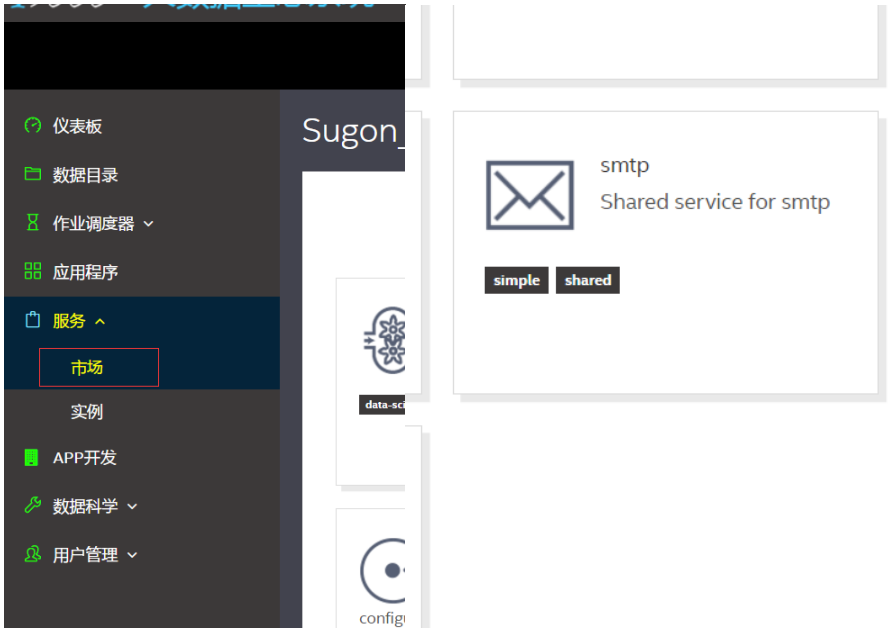
（7）结束此次发送，用 QUIT 命令退出。

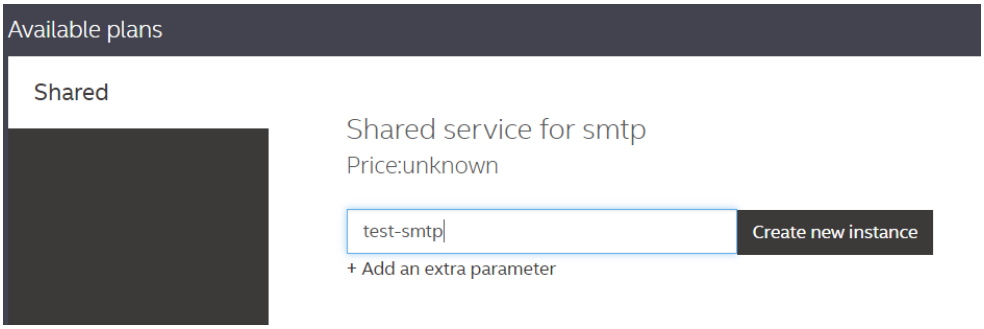
SMTP 服务器基于 DNS 中的邮件交换（MX）记录路由电子邮件。电子邮件系统发邮

件时是根据收信人的地址后缀来定位邮件服务器的。SMTP 通过用户代理程序（UA）完成邮件的编辑、收取和阅读等功能；通过邮件传输代理程序（MTA）将邮件传送到目的地。

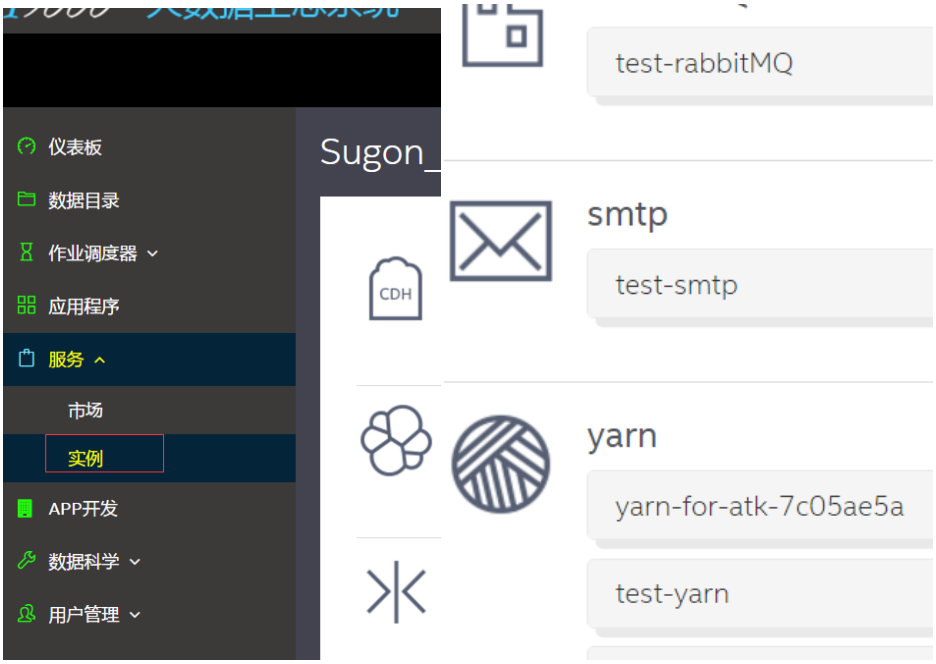
使用方法：

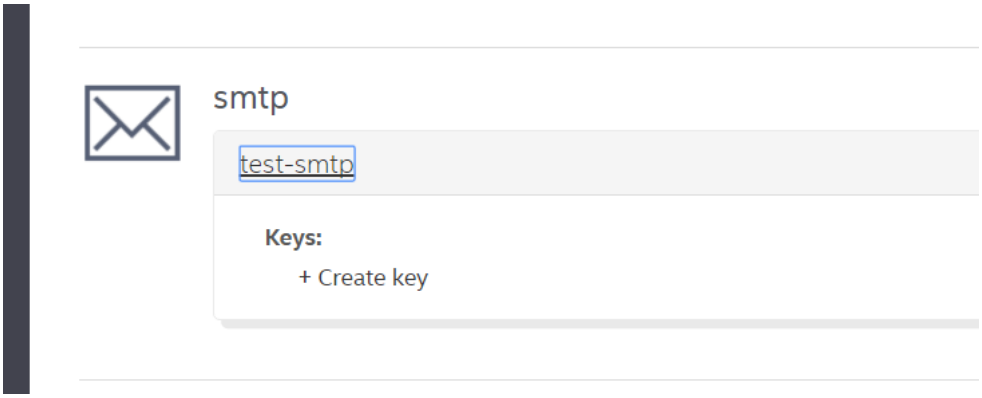
1. 在 i9000 平台上选择 服务>市场，找到 SMTP，点击 SMTP，创建一个新的实例。



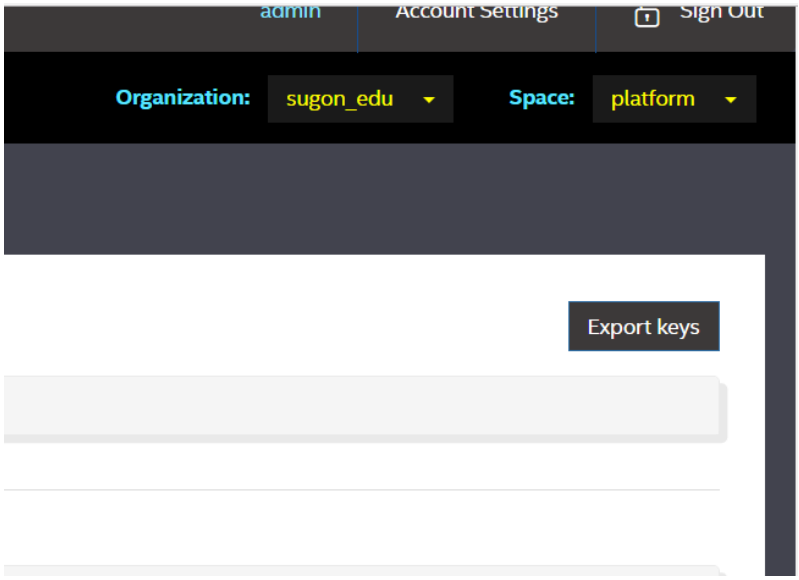


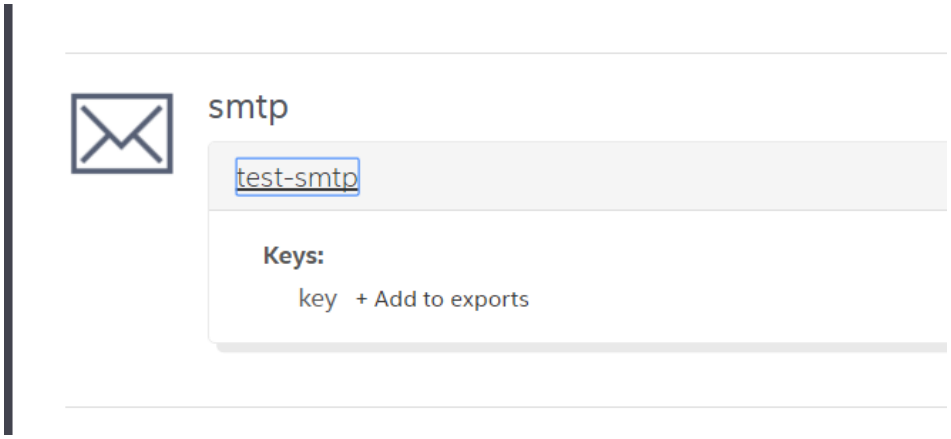
2. 在 服务>实例 里面找到 SMTP，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 SMTP 的连接信息。





exported Keys:


```
{
  "smtp": [
    {
      "label": "smtp",
      "name": "test-smtp",
      "plan": "shared",
      "tags": [
        "simple",
        "shared"
      ],
      "credentials": {
        "host": "smtp.sugonedu.com",
        "password": "Iap123456",
        "port": "25",
        "protocol": "smtp",
        "username": "iap@sugonedu.com"
      }
    }
  ]
}
```

4. 通过这里提供的连接信息可以使用 smtp 实例。

I9000 Analytics Toolkit 概述

大数据对战略洞察的需求推动了大数据分析工具和平台的增长。然而，这些技术的广泛采用面临着许多障碍。成功的采用取决于找到具有大量技能的合适的人，从数据科学专业知识到对大数据平台的深入了解。随着公司的学习，这些技能组需求很高，很难找到。I9000 分析工具包（ATK）是一个可扩展的机器学习框架，通过降低为数据科学家和软件开发人员开发端到端分析的复杂性来解决这些障碍。

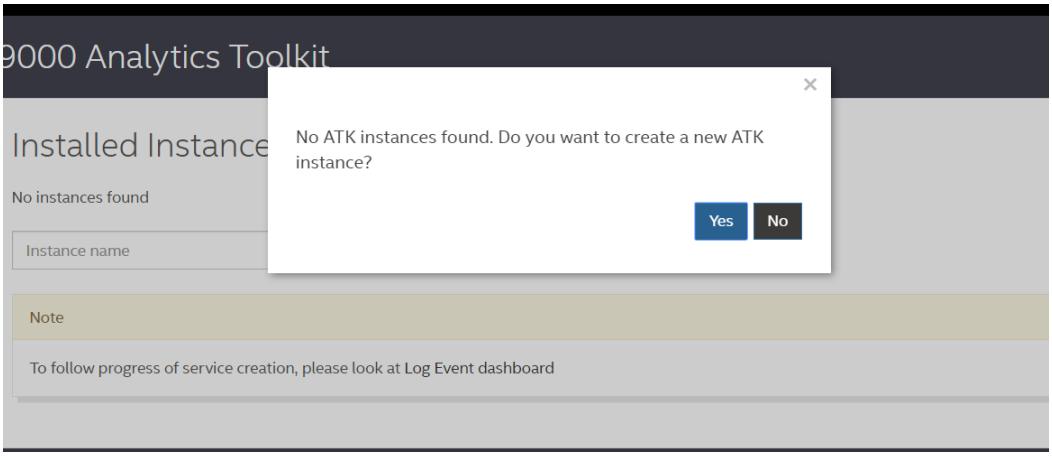
分析工具包可作为 I9000 平台中的应用程序提供。该工具包提供了一个支持批处理和流工作负载的预测分析服务。它允许数据科学家执行大规模数据转换，特征工程，机器学习和图形分析。数据科学家可以使用工具包迭代地构建他们的模型，并且一旦他们的模型准备好生产，就将他们训练的模型发布到评分服务。

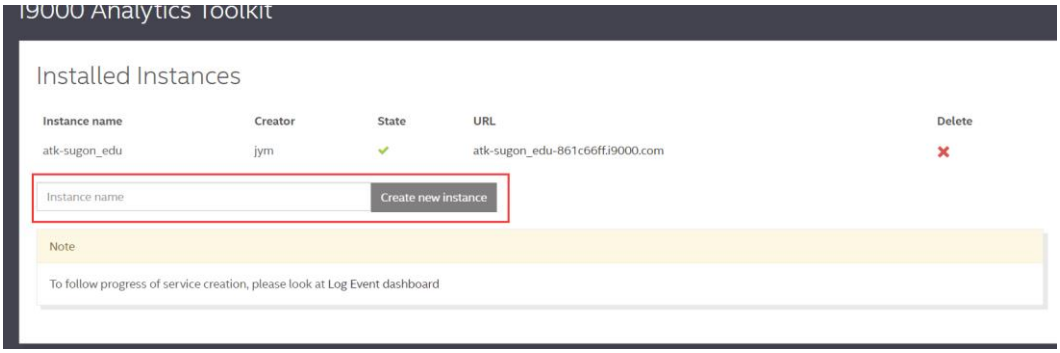
Analytics Toolkit 通过使用熟悉的 Python 数据科学抽象，降低了为数据科学家编写大数据工作流的复杂性。该工具包还通过其插件架构加快了新分析功能的开发。插件允许软件开发人员快速扩展和集成新的算法供数据科学家使用。插件实现为瘦 Scala 包装器，框架自动生成相应的客户端接口。

使用方法：

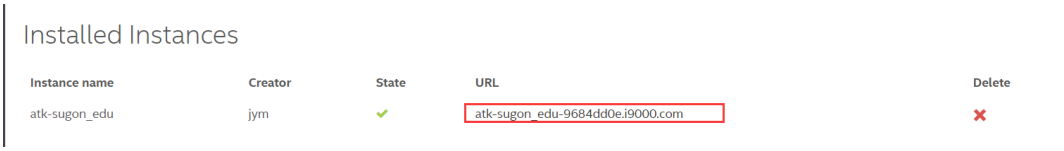
当创建 I9000 Analytics Toolkit 的实例的时候，平台会自动创建与之相绑定的 hbase、hdfs、hive、kerberos、mysql、postgresql、yarn、zookeeper 实例各一个。

1. 在 i9000 平台导航到 数据科学>I9000 数据分析工具，若没有新建过实例则提示是否新建一个实例，选择“Yes”即可新建一个实例。若已经有实例存在，则可以输入实例名称创建一个新实例。





2. 新建的 ATK 实例会提供一个 URL 供使用，可在 Python 代码里使用通过 URL 连接到 ATK REST server 以使用 ATK 工具包对数据进行建模和分析。




```
In [1]: import trustedanalytics as atk
        print "ATK installation path = %s" % (atk.__path__)
        # Replace the server uri value with your ATK instance uri
        atk.server.uri = 'atk-sugon_edu-9684dd0e.i9000.com
```

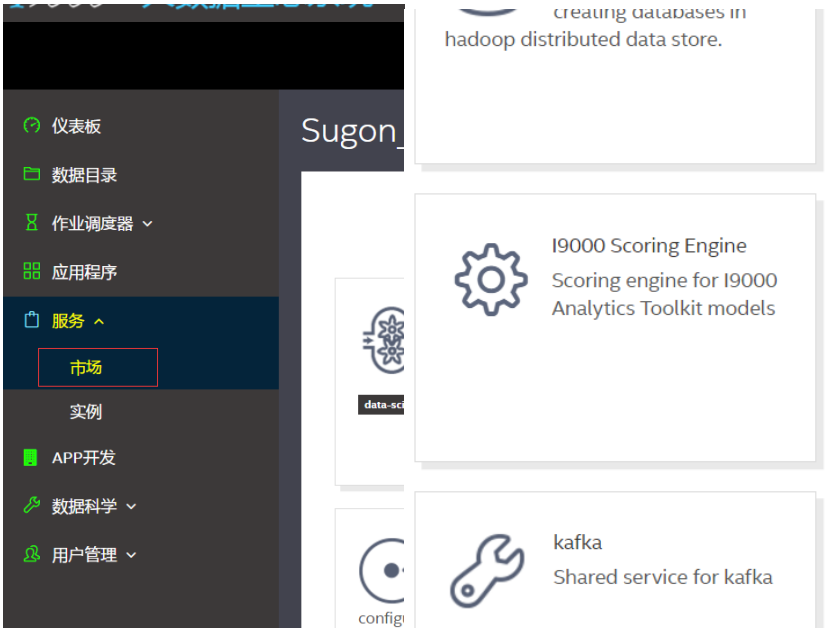
I9000 Scoring Engine 概述

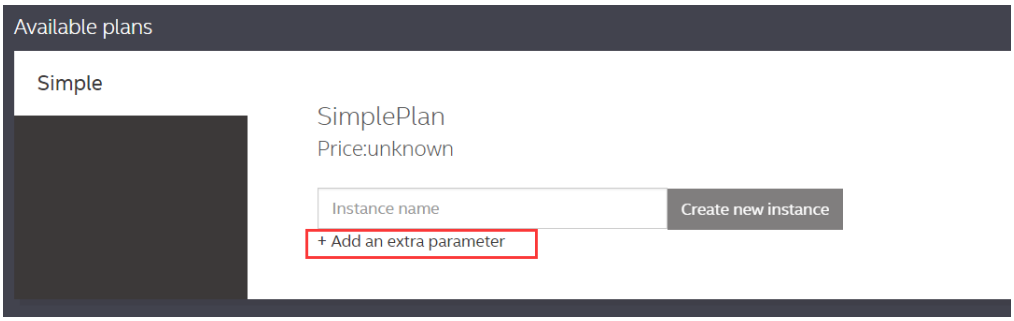
I9000 Scoring Engine 是一个简单的 REST 服务器，能够加载由 ATK 导出的成熟的机器学习模型，并使用模型分析输入数据流。这些模型实现了 Model ARchiver 仓库中定义的 Model ARchive 接口 ,应用程序可以使用评分引擎 RESTful API 来获取模型生成的预测。

I9000 Scoring Engine 支持修改相同类型的模型，并使用相同的 I / O 参数无缝更新，而无需重新部署 I9000 Scoring Engine。它还支持强制使用可能与以前的修订不兼容的修订模型。

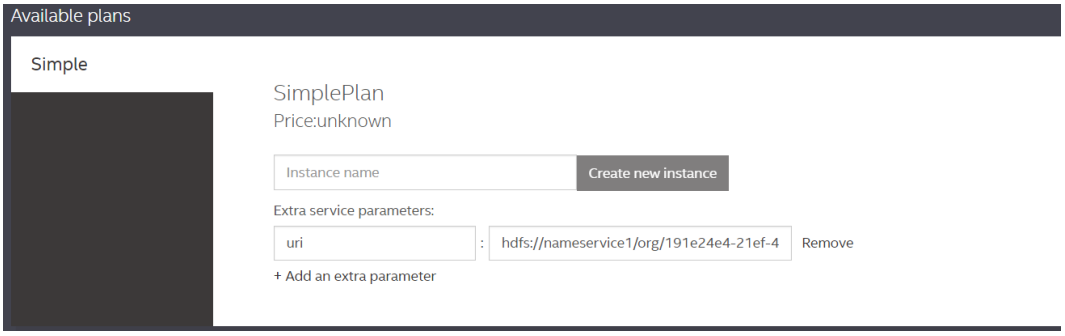
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 I9000 Scoring Engine，点击 I9000 Scoring Engine，创建一个新的实例。

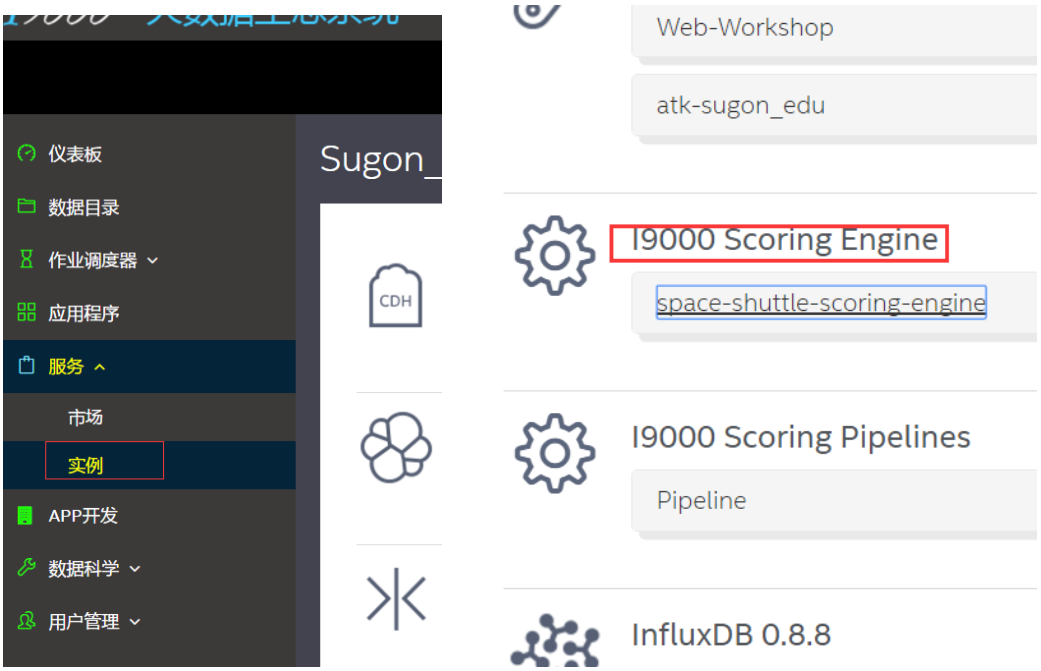


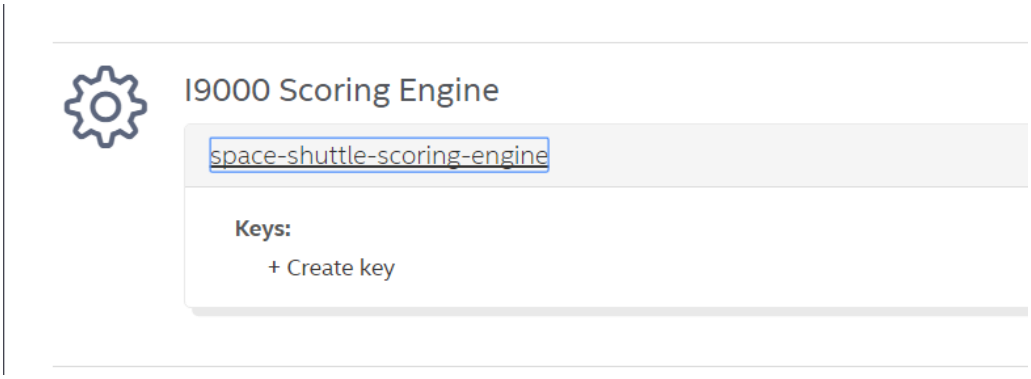


2. 输入实例名字，点击 Add an extra parameter，添加一个要分析的数模型，例如可以添加存储到 hdfs 里的数据模型的 url

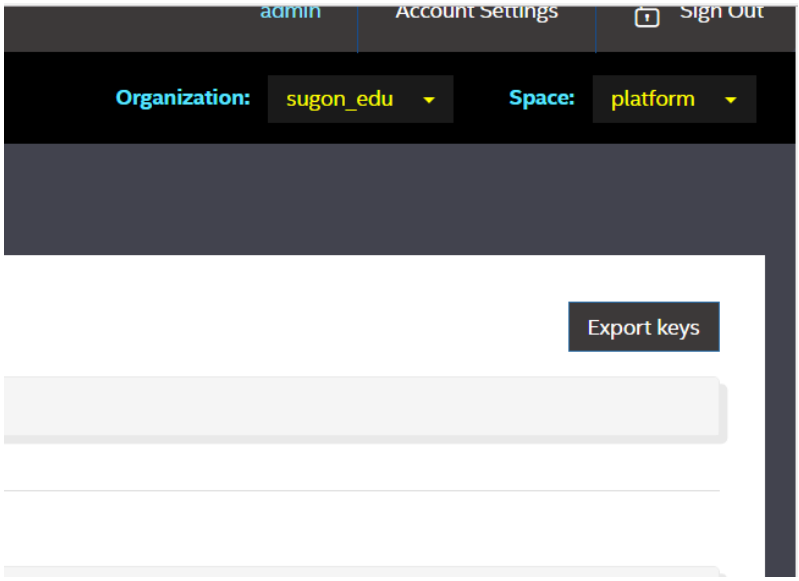


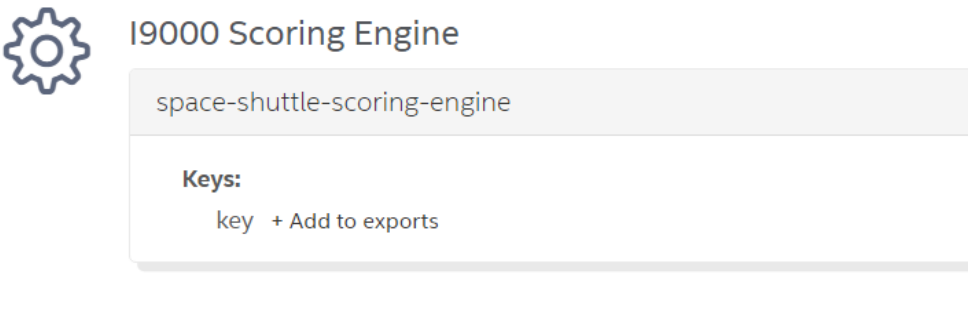
3. 在 服务>实例 里面找到 I9000 Scoring Engine，找到刚刚创建的实例，点击，创建新的 key。





4. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 I9000 Scoring Engine 的连接信息。





exported Keys:


```
{
  "scoring-engine": [
    {
      "label": "scoring-engine",
      "name": "space-shuttle-scoring-engine",
      "plan": "Simple",
      "tags": [],
      "credentials": {
        "url": "space-shuttle-scoring-engine-fd60e8d0.i9000.com"
      }
    }
  ]
}
```


5. 这里将看到新建实例的一个 url，可以在自己的应用程序中使用此 url。

I9000 Scoring Pipelines 概述

I9000 Scoring Pipelines 是一个 Python 应用程序，可用于执行 ETL 转换，然后在部署的模型上（通过 Scoring Engine）记录一行记录。然后，根据应用程序在初始化期间配置的模式，将结果发送回发布请求的客户端或排队等待 Kafka 接收器。

I9000 Scoring Pipelines 目前支持的 ETL 转换是：

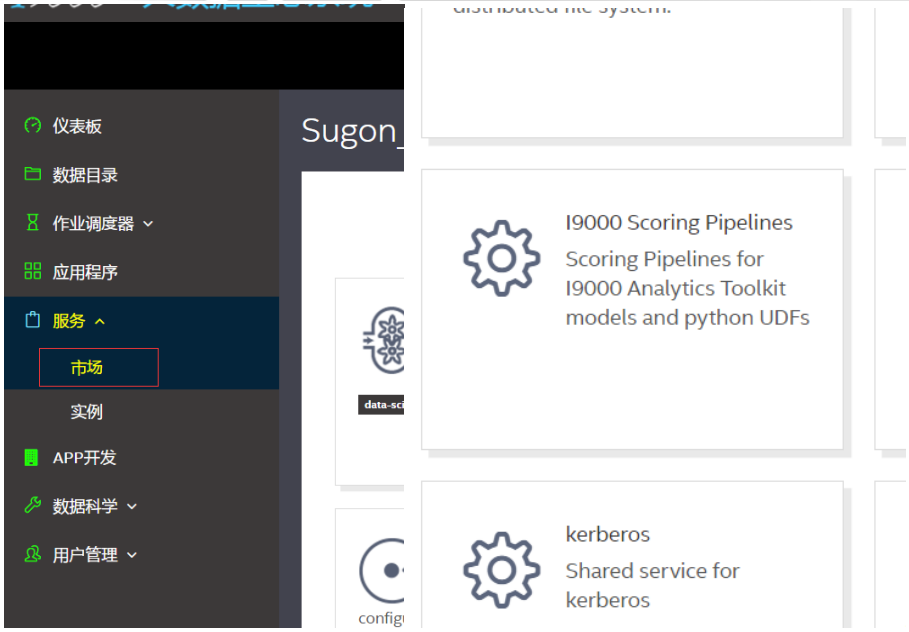
- add_columns
- drop_columns
- rename_columns
- filter
- score

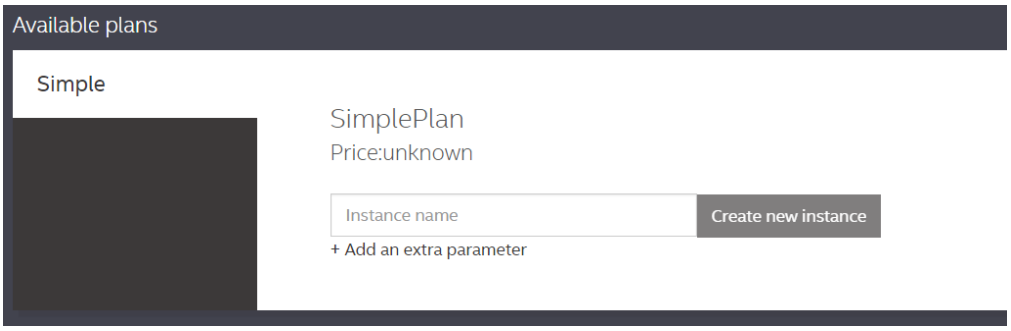
Scoring-engine VS Scoring-pipelines

如果你想要对输入的数据进行转换，那么使用 Scoring-pipelines 而不是 Scoring-engine，Scoring-pipelines 执行支持的数据转换，并自动将输出提交给 Scoring-engine。

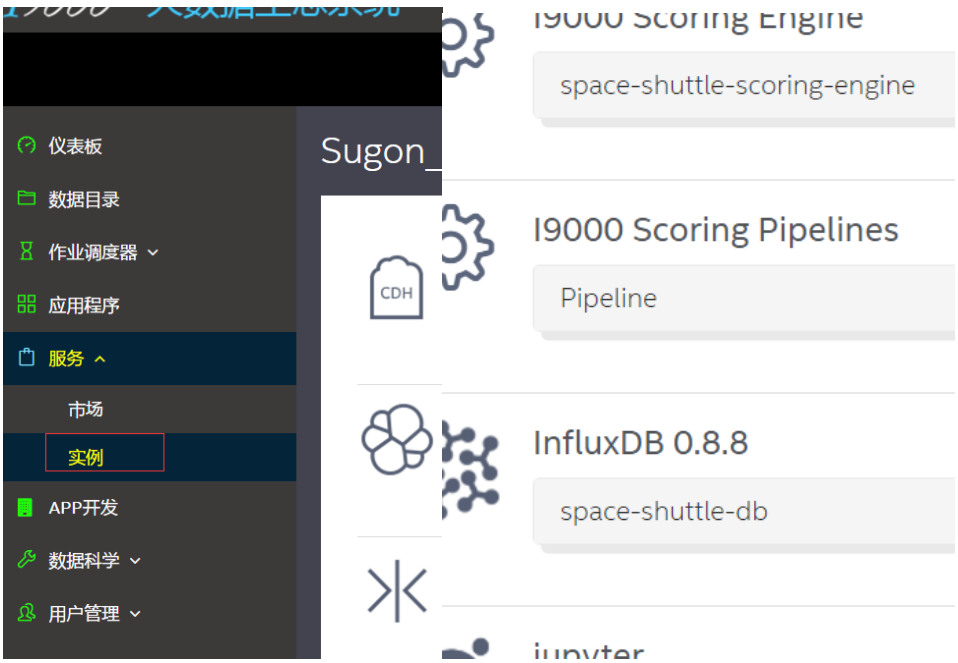
使用方法：

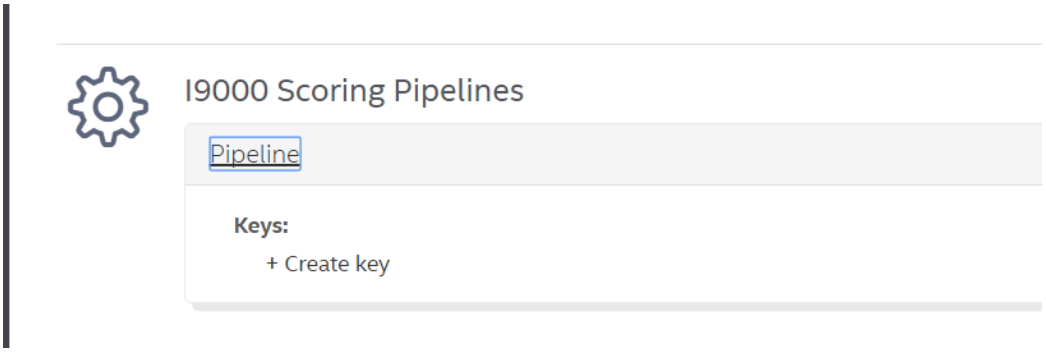
1. 在 i9000 平台上选择 服务>市场 ,找到 I9000 Scoring Pipelines ,点击 I9000 Scoring Pipelines，创建一个新的实例。



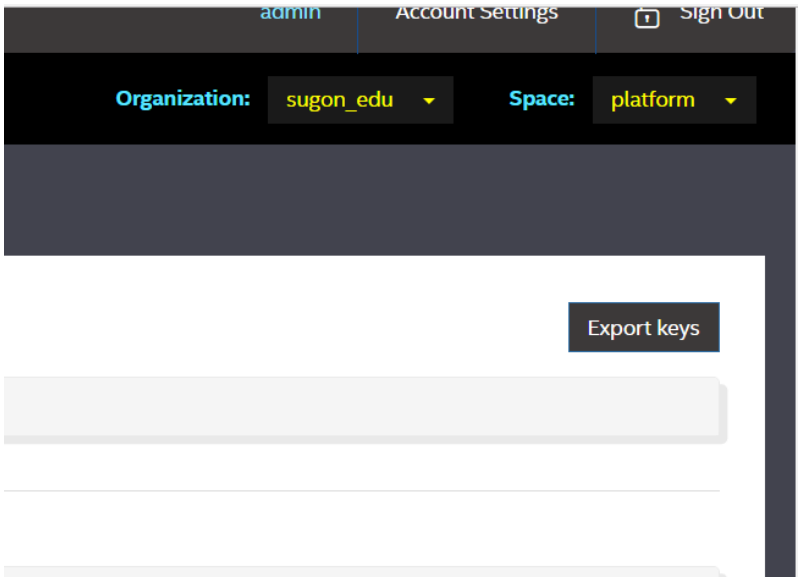


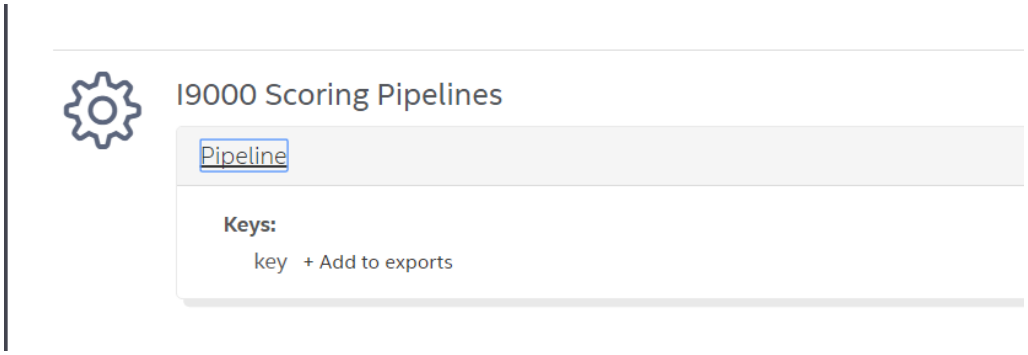
2. 在 服务>实例 里面找到 I9000 Scoring Pipelines，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 I9000 Scoring Pipelines 的连接信息。





3. 这里将看到新建实例的一个 url，可以在自己的应用程序中使用此 url。

Jupyter

概述：

Jupyter Notebook（此前被称为 IPython notebook）是一个交互式笔记本，支持运行 40 多种编程语言 Jupyter Notebook 的本质是一个 Web 应用程序，便于创建和共享文学化程序文档，支持实时代码，数学方程，可视化和 markdown。 用途包括：数据清理和转换，数值模拟，统计建模，机器学习等等[1] 。

用户可以通过电子邮件，Dropbox，GitHub 和 Jupyter Notebook Viewer，将 Jupyter Notebook 分享给其他人。

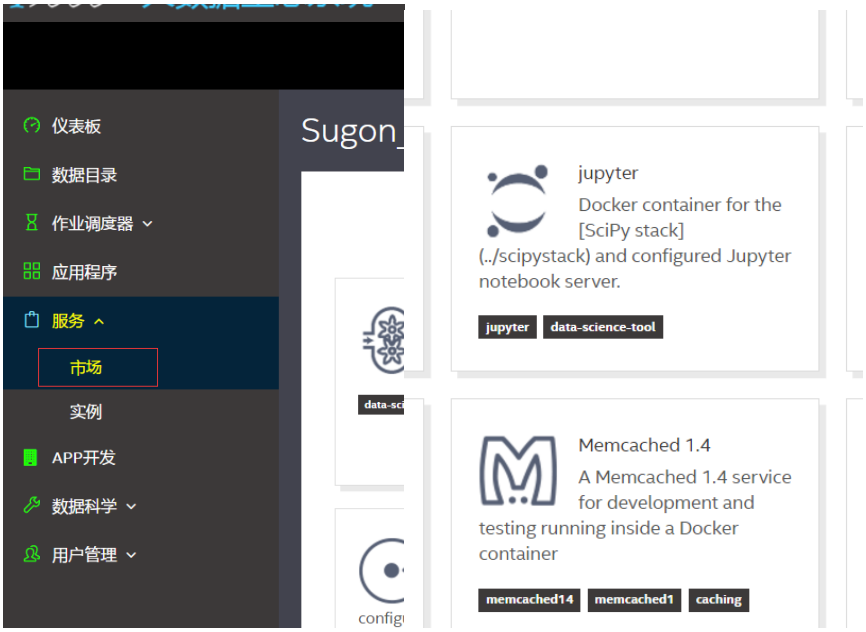
在 Jupyter Notebook 中，代码可以实时的生成图像，视频，LaTeX 和 JavaScript。

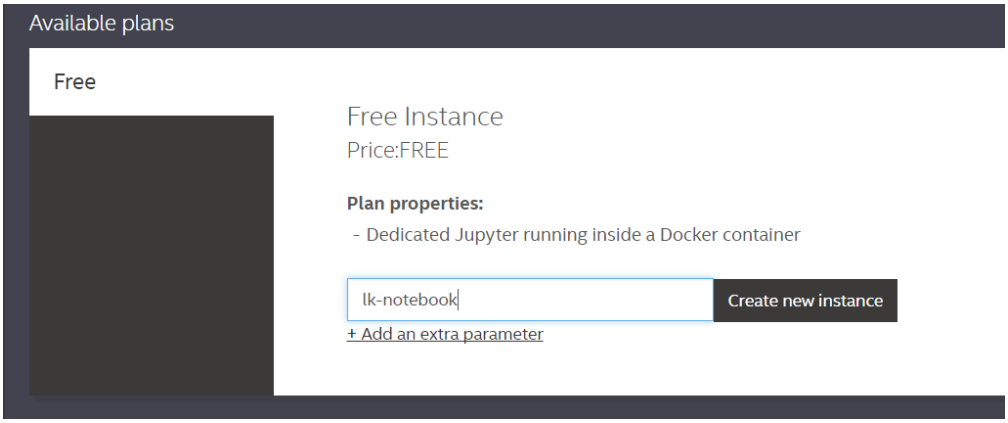
Jupyter 较早的富 GUI 实现应该是 Qt Console。过去在标准 shell 里绘图时，弹出的绘图窗口会接管 shell 会话的控制权，你想继续输入命令就必须先把绘图窗口关掉。这对于希望同时实现可视化和交互式过程的数据分析用户来说显然是难以忍受的，因此 Qt console 站出来解决了这个问题。在 Qt console 中通过 matplotlib 绘制的图形会独立嵌于控制台中，并不影响你继续输入命令。

2011 年，由 Brian Granger 领导的 Jupyter 团队开始开发一种基于 Web 技术的交互式计算文档格式，即 Jupyter notebook。为什么说它是文档格式，而非计算工具呢？实际上它两者都是。Jupyter notebook 在交互上使用了 C/S 结构，它通过 Tornado 建立一个 shell 服务器，并使用浏览器作为客户端。另外 Jupyter notebook 页面都被保存为 .ipynb 的类 JSON 文件格式。这种文件格式也是 Jupyter notebook 最吸引人的地方。

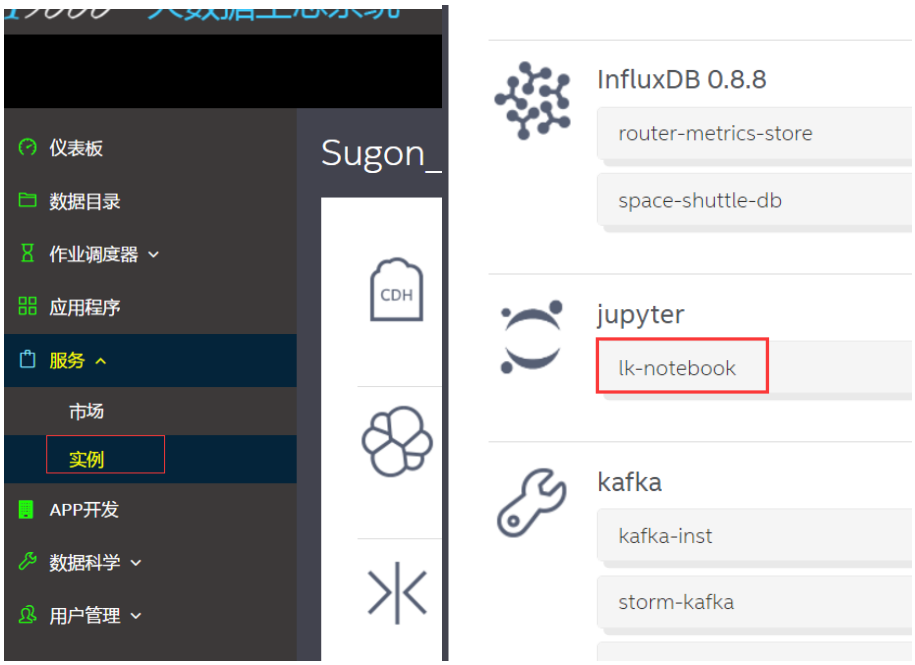
使用方法：

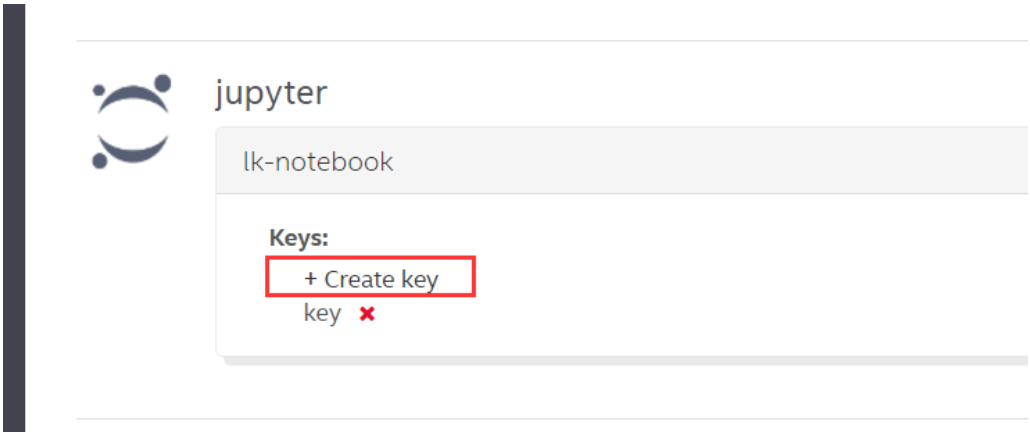
1. 在 i9000 平台上选择 服务>市场，找到 Jupyter，点击 Jupyter，创建一个新的实例。



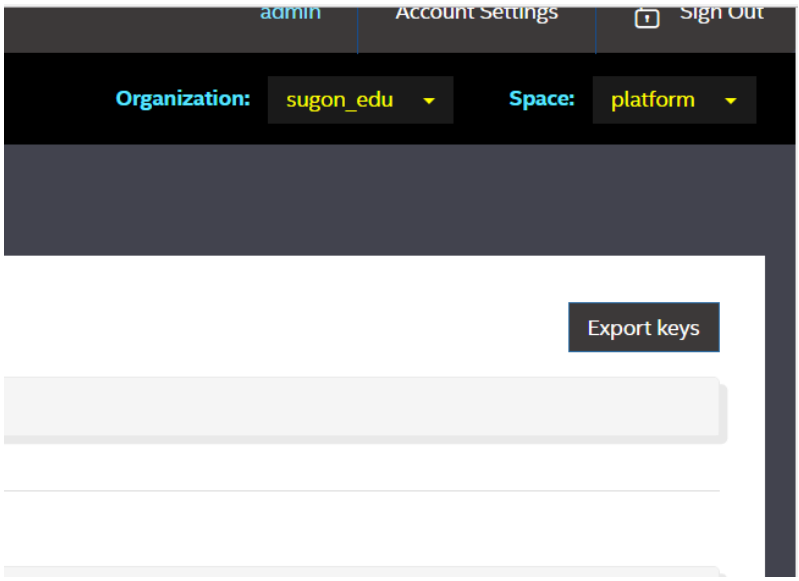


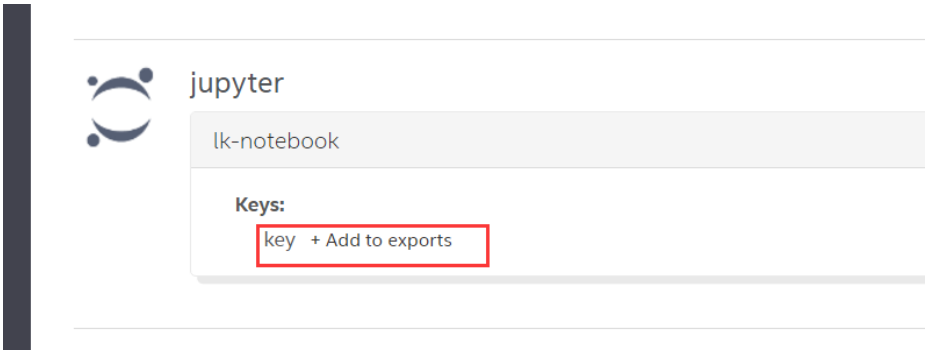
2. 在 服务>实例 里面找到 Jupyter，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Jupyter 的连接信息。



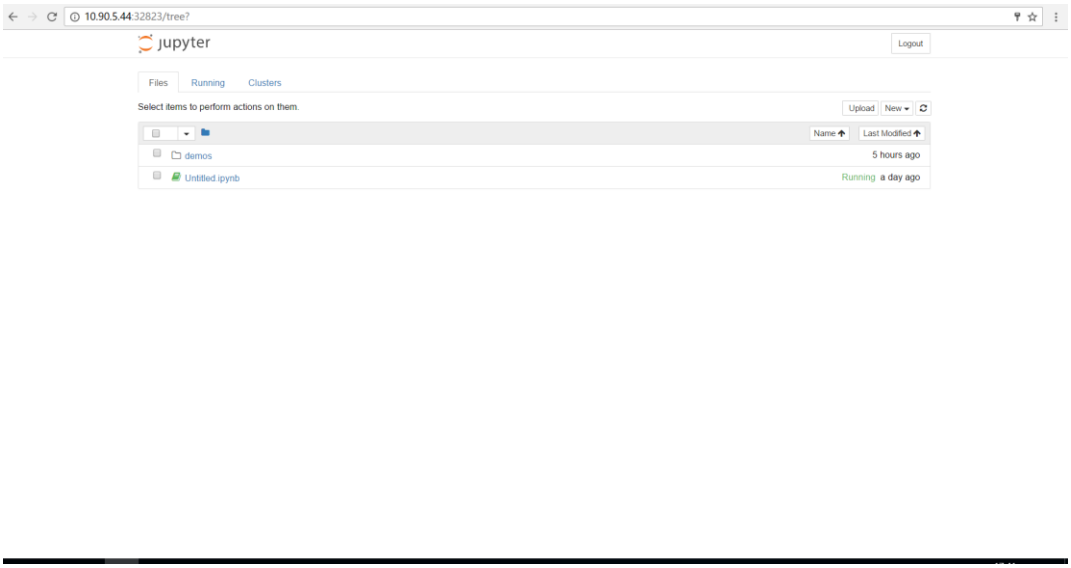


exported Keys:


```
{
  "jupyter": [
    {
      "label": "jupyter",
      "name": "lk-notebook",
      "plan": "free",
      "tags": [
        "jupyter",
        "data-science-tool"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "8888/tcp": "32823"
        },
        "port": "32823",
        "password": "l9hcft1bfqzuqv37",
        "uri": "https://10.0.4.4:32823"
      }
    }
  ]
}
```


4. 在浏览器里输入 uri，可打开刚刚新建的 Jupyter 实例，输入 password 即可使用。


10.90.5.44:32823/login
jupyter
Password:
Log in


10.90.5.44:32823/tree?
jupyter
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
Name
Last Modified
demos
5 hours ago
Untitled.ipynb
Running
a day ago

Kafka

概述：

Kafka 是分布式发布-订阅消息系统。它最初由 LinkedIn 公司开发，之后成为 Apache 项目的一部分。Kafka 是一个分布式的，可划分的，冗余备份的持久性的日志服务。它主要用于处理活跃的流式数据。

在大数据系统中，常常会碰到一个问题，整个大数据是由各个子系统组成，数据需要在各个子系统中高性能，低延迟的不停流转。传统的企业消息系统并不是非常适合大规模的数据处理。为了已在同时搞定在线应用（消息）和离线应用（数据文件，日志）Kafka 就出现了。Kafka 可以起到两个作用：

1.降低系统组网复杂度。

2.降低编程复杂度，各个子系统不在是相互协商接口，各个子系统类似插口插在插座上，Kafka 承担高速数据总线的作用。

Kafka 主要特点：

1.同时为发布和订阅提供高吞吐量。据了解，Kafka 每秒可以生产约 25 万消息（50 MB），每秒处理 55 万消息（110 MB）。

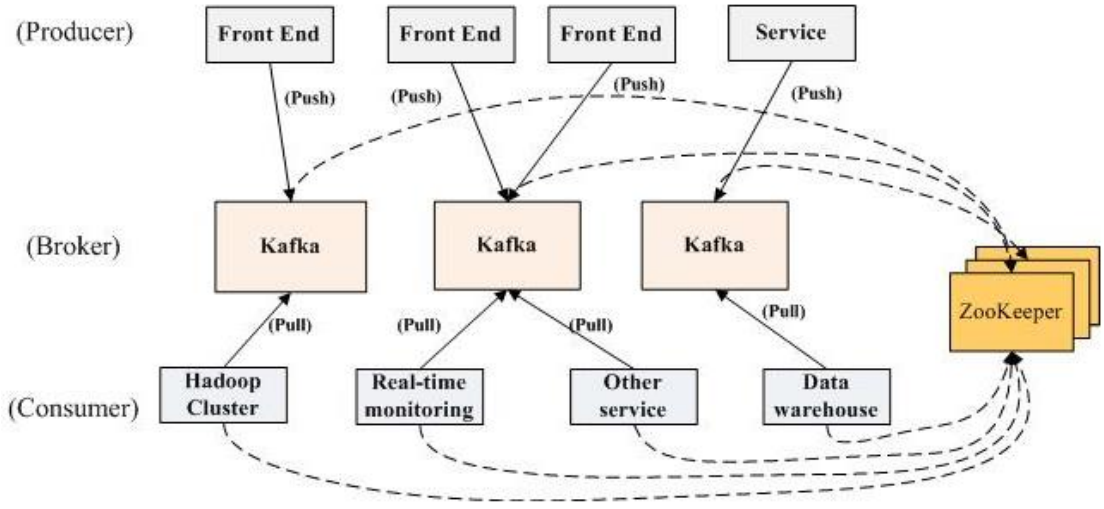
2.可进行持久化操作。将消息持久化到磁盘，因此可用于批量消费，例如 ETL，以及实时应用程序。通过将数据持久化到硬盘以及 replication 防止数据丢失。

3.分布式系统，易于向外扩展。所有的 producer、broker 和 consumer 都会有多个，均为分布式的。无需停机即可扩展机器。

4.消息被处理的状态是在 consumer 端维护，而不是由 server 端维护。当失败时能自动平衡。

5.支持 online 和 offline 的场景。

Kafka 的架构：



Kafka 的整体架构非常简单，是显式分布式架构，producer、broker（kafka）和 consumer 都可以有多个。Producer ,consumer 实现 Kafka 注册的接口 数据从 producer 发送到 broker，broker 承担一个中间缓存和分发的作用。broker 分发注册到系统中的 consumer。broker 的作用类似于缓存，即活跃的数据和离线处理系统之间的缓存。客户端和服务器端的通信，是基于简单，高性能，且与编程语言无关的 TCP 协议。几个基本概念：

1.Topic：特指 Kafka 处理的消息源（feeds of messages）的不同分类。

2.Partition：Topic 物理上的分组，一个 topic 可以分为多个 partition，每 partition 是一个有序的队列。partition 中的每条消息都会被分配一个有序的 id（offset）。

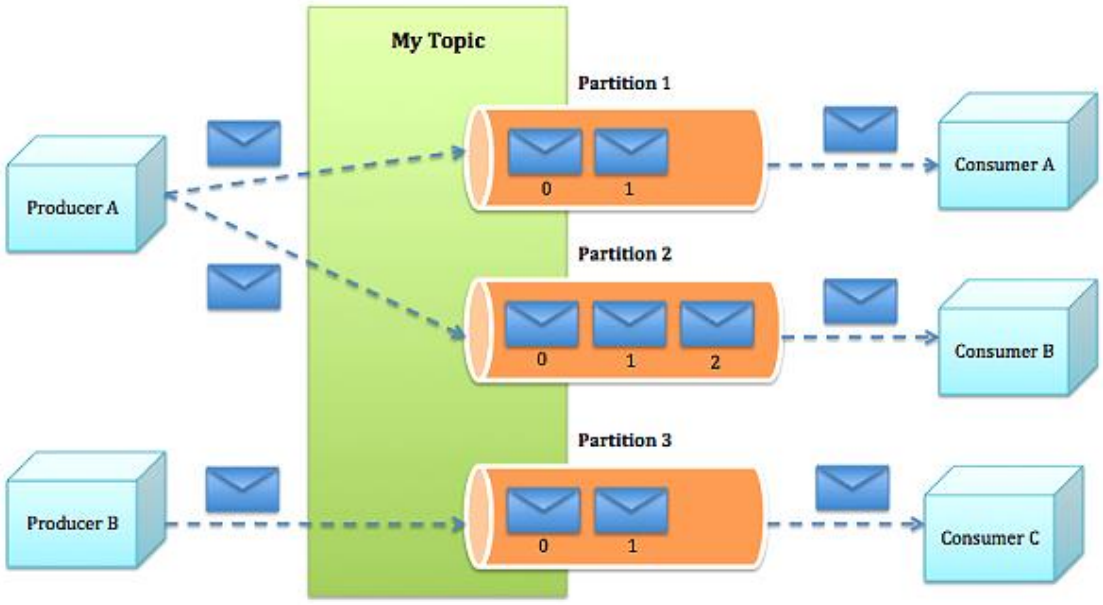
3.Message：消息，是通信的基本单位，每个 producer 可以向一个 topic（主题）发布一些消息。

4.Producers：消息和数据生产者，向 Kafka 的一个 topic 发布消息的过程叫做 producers。

5.Consumers：消息和数据消费者，订阅 topics 并处理其发布的消息的过程叫做 consumers。

6.Broker：缓存代理，Kafa 集群中的一台或多台服务器统称为 broker。

消息发送的流程：



1.Producer 根据指定的 partition 方法（round-robin、hash 等），将消息发布到指定 topic 的 partition 里面

2.kafka 集群接收到 Producer 发过来的消息后，将其持久化到硬盘，并保留消息指定时长（可配置），而不关注消息是否被消费。

3.Consumer 从 kafka 集群 pull 数据，并控制获取消息的 offset。

Kafka 的设计要点：

1、直接使用 linux 文件系统的 cache，来高效缓存数据。

2、采用 linux Zero-Copy 提高发送性能。传统的数据发送需要发送 4 次上下文切换，采用 sendfile 系统调用之后，数据直接在内核态交换，系统上下文切换减少为 2 次。根据测试结果，可以提高 60%的数据发送性能。

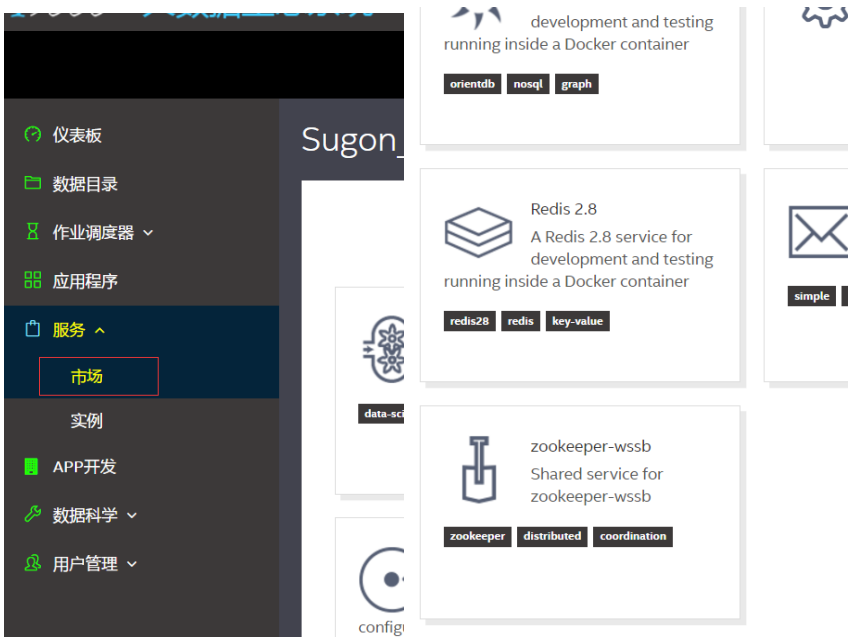
3、数据在磁盘上存取代价为 O(1)。kafka 以 topic 来进行消息管理，每个 topic 包含多个 part（ition），每个 part 对应一个逻辑 log，有多个 segment 组成。每个 segment 中存储多条消息（见下图），消息 id 由其逻辑位置决定，即从消息 id 可直接定位到消息的存储位置，避免 id 到位置的额外映射。每个 part 在内存中对应一个 index，记录每个 segment 中的第一条消息偏移。发布者发到某个 topic 的消息会被均匀的分布到多个 part 上（随机或根据用户指定的回调函数进行分布），broker 收到发布消息往对应 part 的最后一个 segment 上添加该消息，当某个 segment 上的消息条数达到配置值或消息发布时间超过阈值时，segment 上的消息会被 flush 到磁盘，只有 flush 到磁盘上的消息订阅者才能订阅到，segment 达到一定的大小后将不会再往该 segment 写数据，broker 会创建新的 segment。

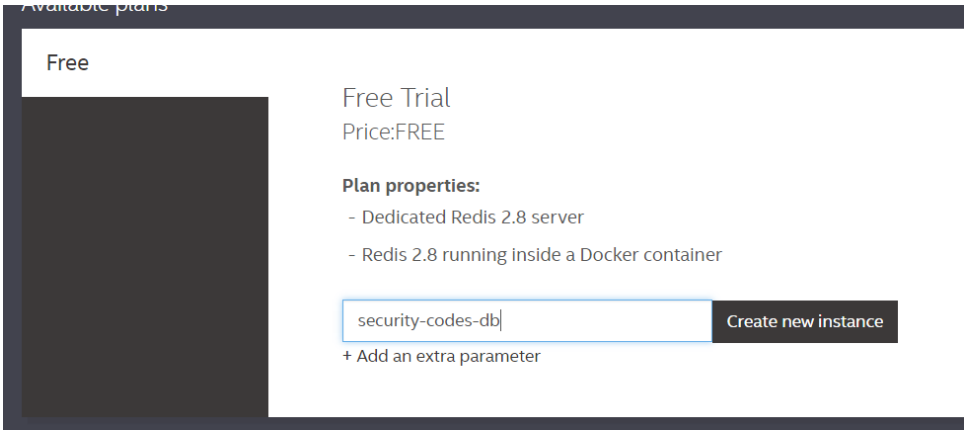
4、显式分布式，即所有的 producer、broker 和 consumer 都会有多个，均为分布式的。Producer 和 broker 之间没有负载均衡机制。broker 和 consumer 之间利用 zookeeper 进行负载均衡。所有 broker 和 consumer 都会在 zookeeper 中进行注册，且 zookeeper 会保存他们的一些元数据信息。如果某个 broker 和 consumer 发生了变化，所有其他的

broker 和 consumer 都会得到通知。

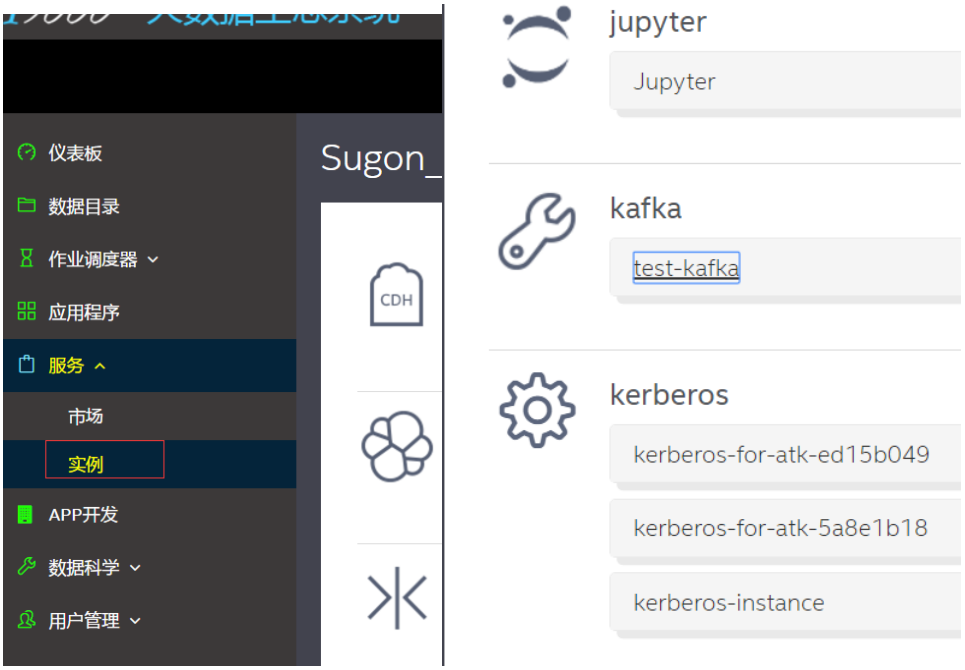
使用方法：

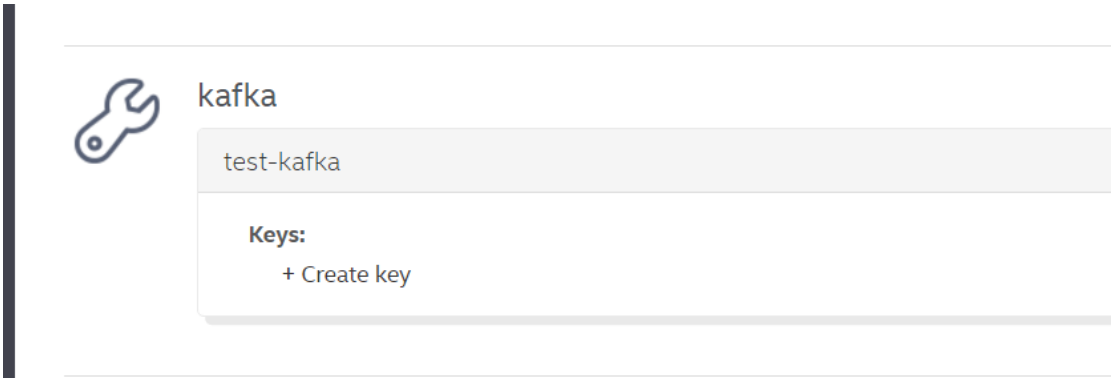
1. 在 i9000 平台上选择 服务>市场，找到 Kafka，点击 Kafka，创建一个新的实例。



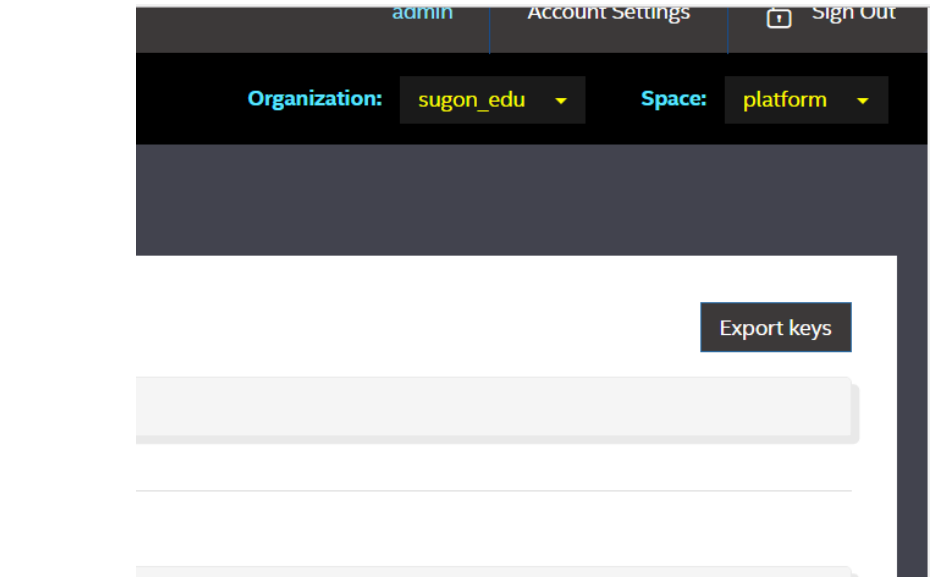


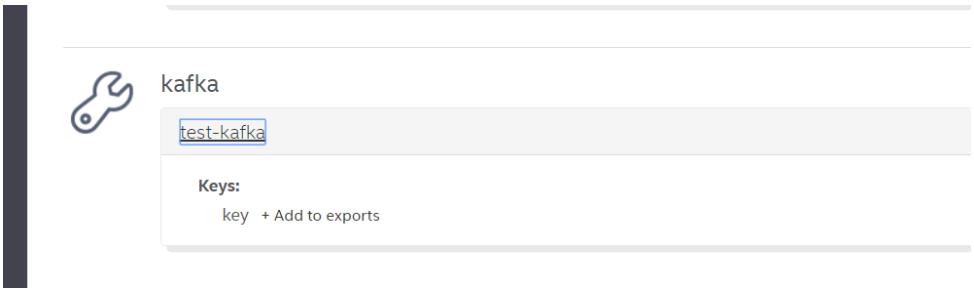
2. 在 服务>实例 里面找到 Kafka，找到刚刚创建的实例，点击，创建新的 key。

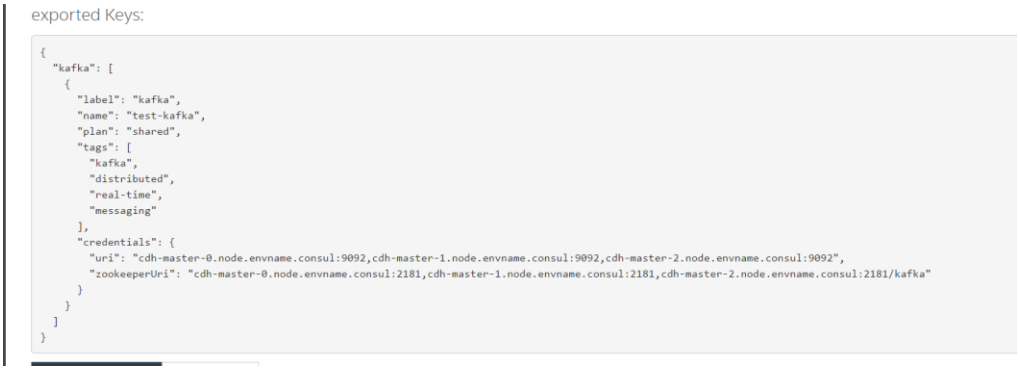




3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Kafka 的配置信息。







4. 这里提供了三个节点的 uri 以及 zookeeperUri，可使用这些配置 kafka。

Kerberos

概述：

Kerberos 是一种网络认证协议，其设计目标是通过密钥系统为客户机 / 服务器应用程序提供强大的认证服务。该认证过程的实现不依赖于主机操作系统的认证，无需基于主机地址的信任，不要求网络上所有主机的物理安全，并假定网络上传送的数据包可以被任意地读取、修改和插入数据。在以上情况下， Kerberos 作为一种可信任的第三方认证服务，是通过传统的密码技术（如：共享密钥）执行认证服务的。

认证过程具体如下：客户机向认证服务器（AS）发送请求，要求得到某服务器的证书，然后 AS 的响应包含这些用客户端密钥加密的证书。证书的构成为： 1）服务器“ticket” ；2) 一个临时加密密钥(又称为会话密钥 “session key” ）。客户机将 ticket（包括用服务器密钥加密的客户机身份和一份会话密钥的拷贝）传送到服务器上。会话密钥可以（现已经由客户机和服务器共享）用来认证客户机或认证服务器，也可用来为通信双方以后的通讯提供加密服务 ,或通过交换独立子会话密钥为通信双方提供进一步的通信加密服务。

上述认证交换过程需要只读方式访问 Kerberos 数据库。但有时，数据库中的记录必须进行修改，如添加新的规则或改变规则密钥时。修改过程通过客户机和第三方 Kerberos 服务器（Kerberos 管理器 KADM）间的协议完成。有关管理协议在此不作介绍。另外也有一种协议用于维护多份 Kerberos 数据库的拷贝，这可以认为是执行过程中的细节问题，并且会不断改变以适应各种不同数据库技术。

Kerberos 又指麻省理工学院为这个协议开发的一套计算机网络安全系统。系统设计上采用客户端/服务器结构与 DES 加密技术，并且能够进行相互认证，即客户端和服务器端均

可对对方进行身份认证。可以用于防止窃听、防止 replay 攻击、保护数据完整性等场合，是一种应用对称密钥体制进行密钥管理的系统。Kerberos 的扩展产品也使用公开密钥加密方法进行认证。

工作原理：

简要大概地说一下 Kerberos 是如何工作的：

假设你要在一台电脑上访问另一个服务器（你可以发送 telnet 或类似的登录请求）。你知道服务器要接受你的请求必须要有一张 Kerberos 的“入场券”。

要得到这张入场券，你首先要向验证服务器（AS）请求验证。验证服务器会创建基于你的密码（从你的用户名而来）的一个“会话密钥”（就是一个加密密钥），并产生一个代表请求的服务的随机值。这个会话密钥就是“允许入场的入场券”。

然后，你把这张允许入场的入场券发到授权服务器（TGS）。TGS 物理上可以和验证服务器是同一个服务器，只不过它现在执行的是另一个服务。TGS 返回一张可以发送给请求服务的服务器的票据。

服务器或者拒绝这张票据，或者接受这张票据并执行服务。

因为你从 TGS 收到的这张票据是打上时间戳的，所以它允许你在某个特定时期内（一般是八小时）不用再验证就可以使用同一张票来发出附加的请求。使这张票拥有一个有限的有效期使其以后不太可能被其他人使用。

实际的过程要比刚才描述的复杂得多。用户过程也会根据具体执行有一些改变。

缺陷：

1.失败于单点：它需要中心服务器的持续响应。当 Kerberos 服务结束前，没有人可以连接到服务器。这个缺陷可以通过使用复合 Kerberos 服务器和缺陷认证机制弥补。

2.Kerberos 要求参与通信的主机的时钟同步。票据具有一定有效期，因此，如果主机的时钟与 Kerberos 服务器的时钟不同步，认证会失败。默认设置要求时钟的时间相差不超过 10 分钟。在实践中，通常用网络时间协议后台程序来保持主机时钟同步。

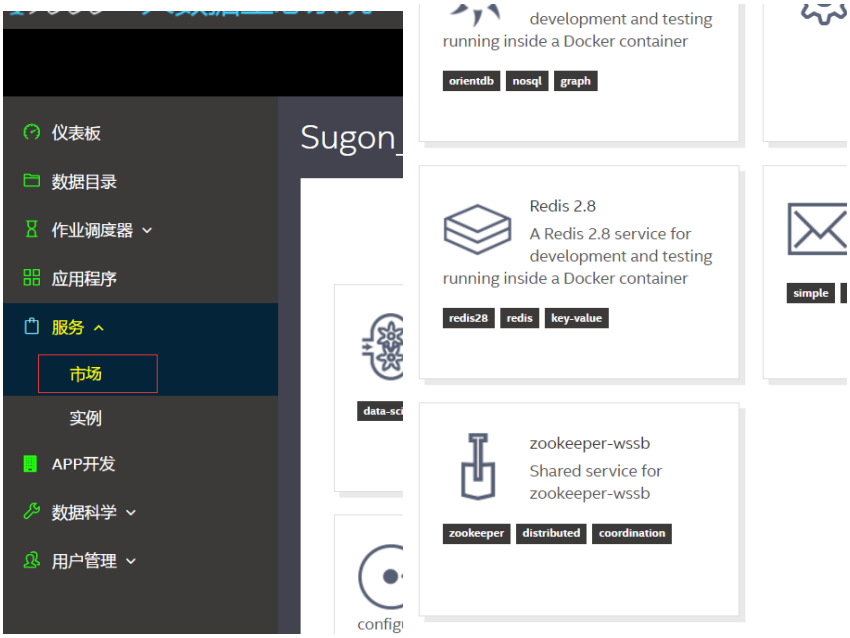
3.管理协议并没有标准化，在服务器实现工具中有一些差别。

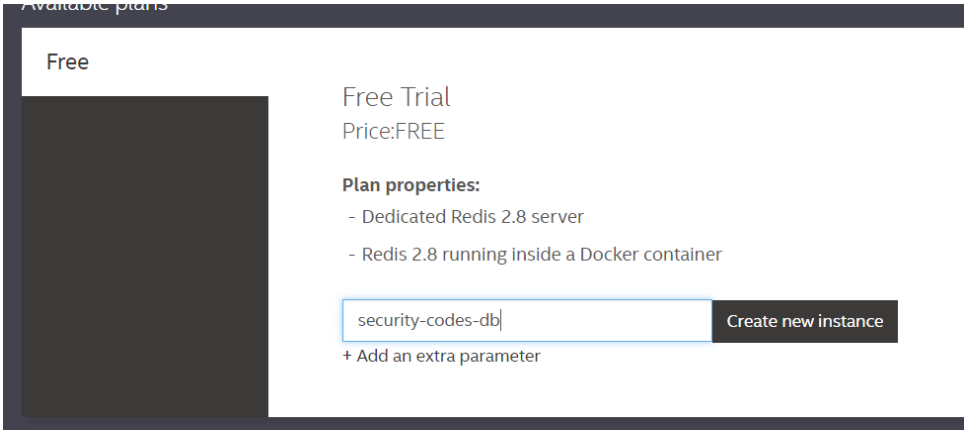
4.因为所有用户使用的密钥都存储于中心服务器中，危及服务器的安全的行为将危及所有用户的密钥。

5.一个危险客户机将危及用户密码。

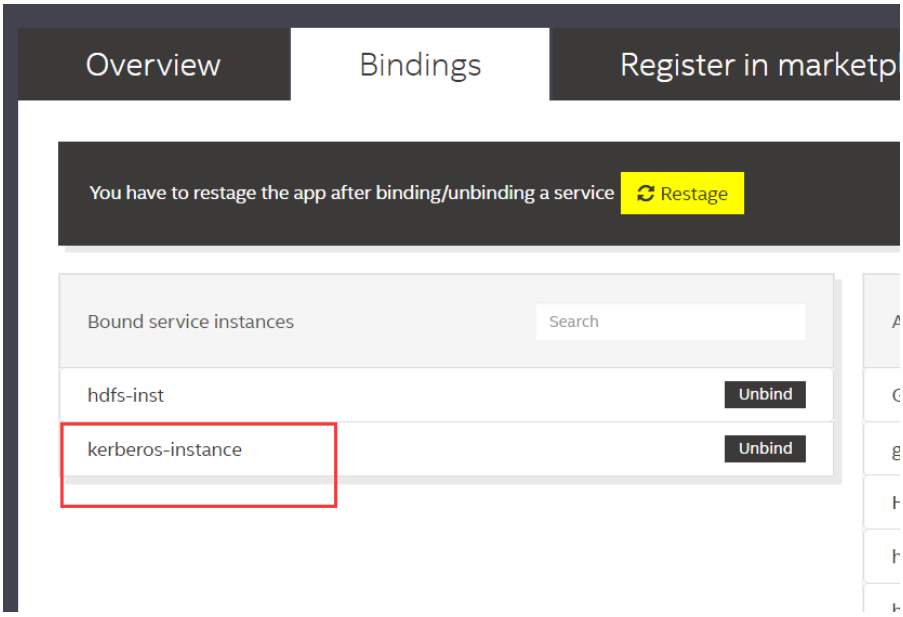
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 Kerberos，点击 Kerberos，创建一个新的实例。

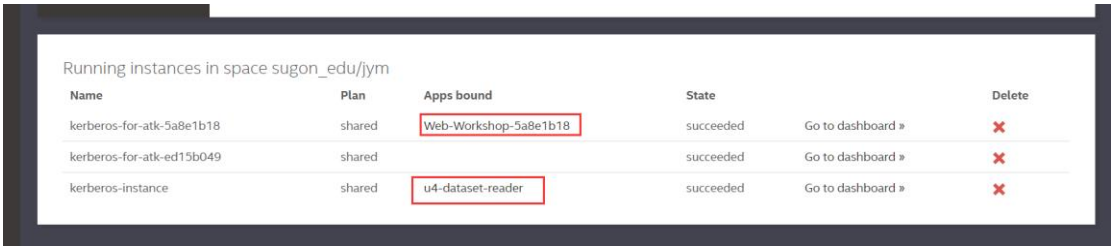




2. 创建后即可在下面看到刚刚创建的实例，在其他实例或者自己发布应用的时候可以通过实例的名字绑定该实例。



3. 被绑定的实例会显示自己被哪个应用绑定了。



logstash 1.4

简介：

Logstash 是一个完全开源的工具，他可以对你的日志进行收集、分析，并将其存储供以后使用（如，搜索）。说到搜索，logstash 带有一个 web 界面，搜索和展示所有日志。

Logstash 经常与 ElasticSearch，Kibana 配置，组成著名的 ELK 技术栈，非常适合用来做日志数据的分析。当然它可以单独出现，作为日志收集软件，你可以收集日志到多种存储系统或临时中转系统，如 MySQL，redis，kakfa，HDFS，lucene，solr 等并不一定是 ElasticSearch。

Inputs,Outputs,Codecs,Filters 构成了 Logstash 的核心配置项。Logstash 通过建立一条事件处理的管道，从你的日志提取出数据保存到 Elasticsearch 中，为高效的查询数据提供基础。

数据在许多格式的许多系统中经常是分散的或孤立的。Logstash 支持各种各样的输入，它们可以同时从大量的常见源引发事件。从您的日志，指标，网络应用程序，数据存储和各种 AWS 服务中轻松获取所有这些服务，都以连续流式传输方式。

虽然 Elasticsearch 的开放式输出，打开了一个搜索和分析能力的新世界，但它不是唯一可用的。Logstash 具有多种输出，可让您将数据路由到所需的位置，使您可以灵活地解锁大量下游用例。

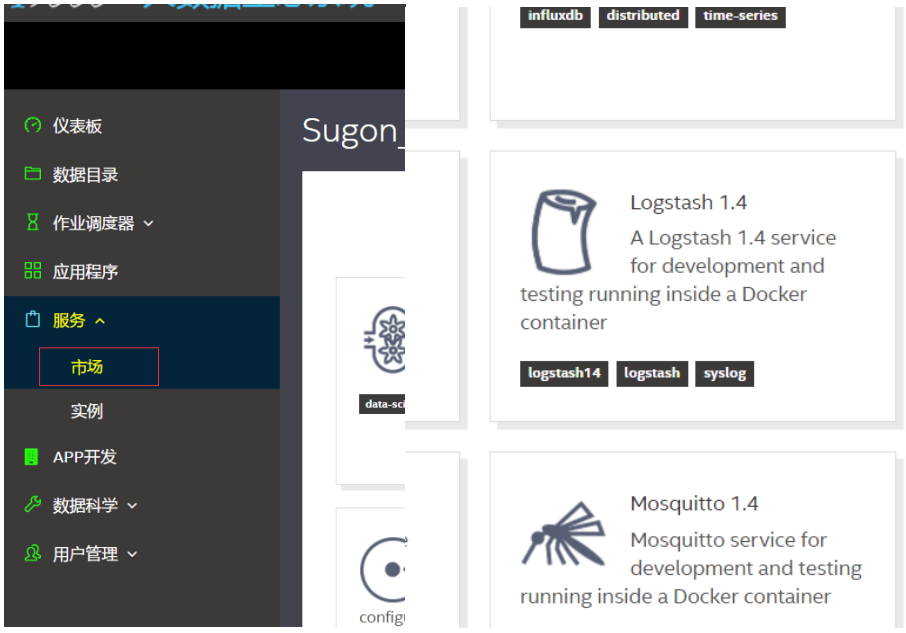
随着数据从源代码到存储，Logstash 过滤器解析每个事件，识别命名字段以构建结构，并将其转换为以通用格式收敛以实现更简单，更快速的分析和业务价值。

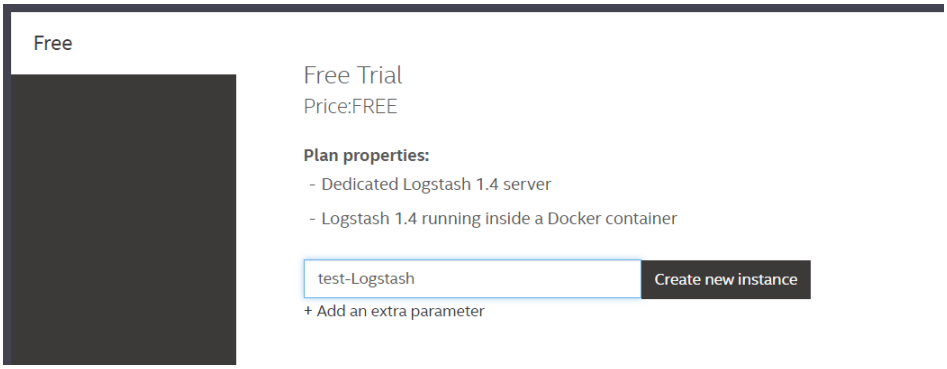
Logstash 动态转换和准备您的数据，无论格式或复杂性如何：

- 用 grok 从非结构化数据导出结构
- 从 IP 地址解密地理坐标
- 匿名 PII 数据，完全排除敏感字段
- 简化整体处理，独立于数据源，格式或模式。

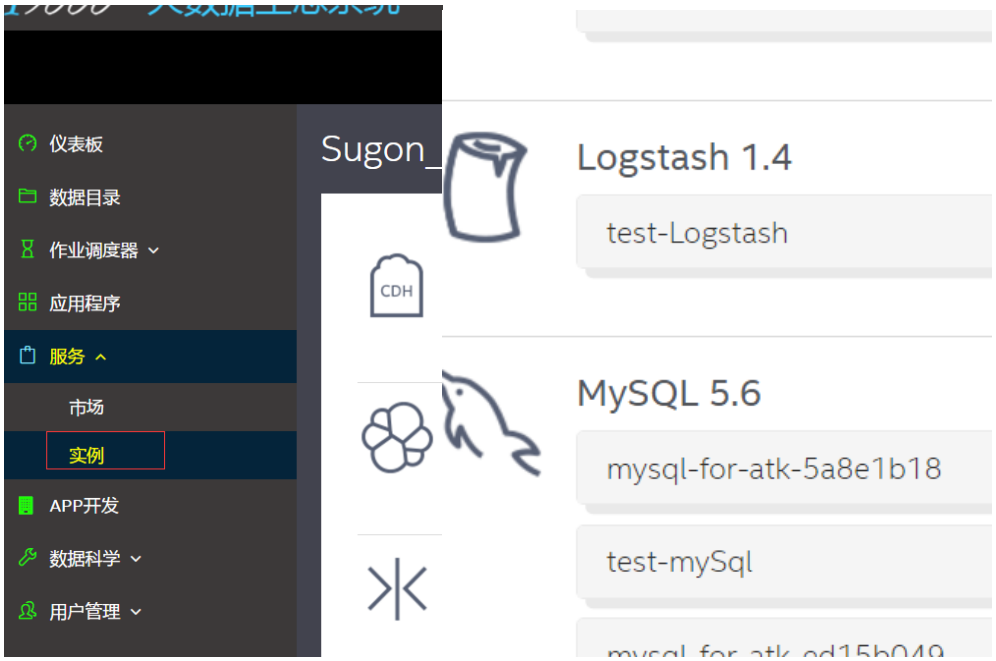
使用方法：

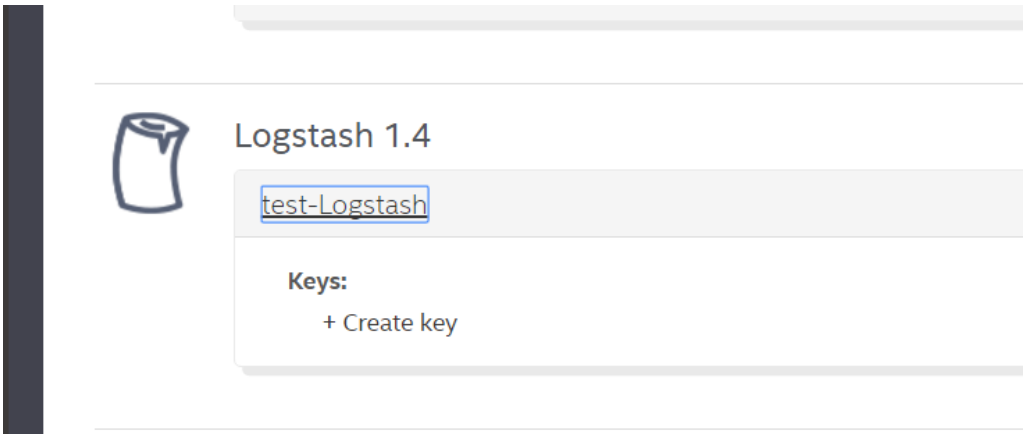
1. 在 i9000 平台上选择 服务>市场，找到 Logstash，点击 Logstash，创建一个新的实例。



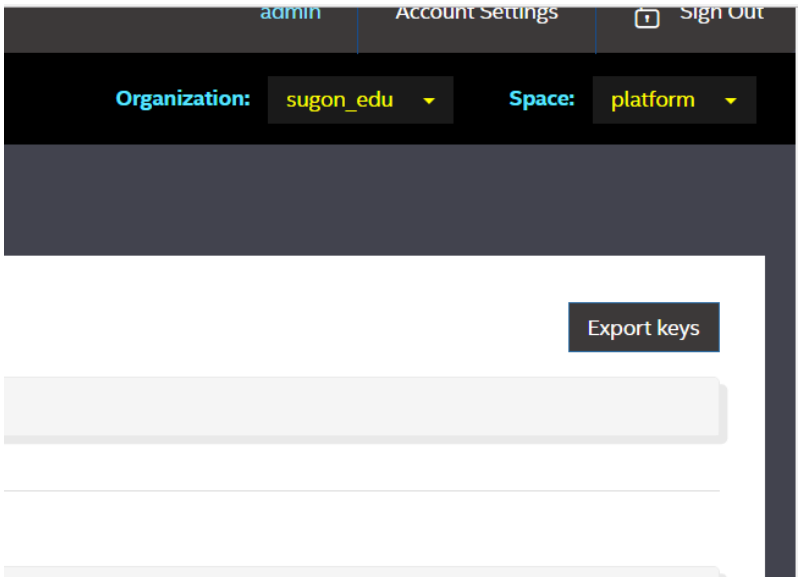


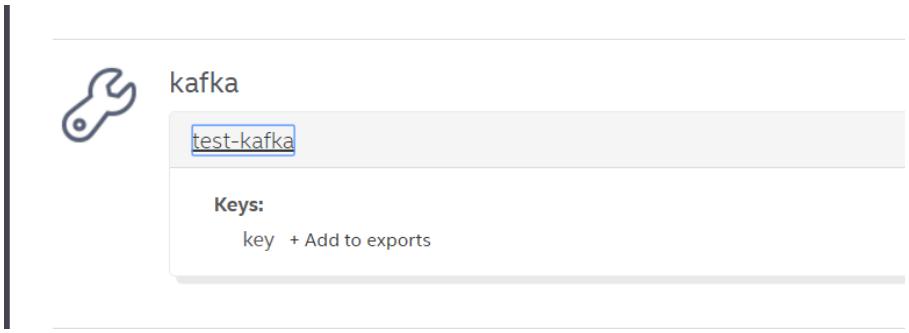
2. 在 服务>实例 里面找到 Logstash，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Logstash 的配置信息。





exported Keys:


```
{
  "logstash14": [
    {
      "label": "logstash14",
      "name": "test-Logstash",
      "plan": "free",
      "tags": [
        "logstash14",
        "logstash",
        "syslog"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "514/tcp": "32893",
          "9200/tcp": "32894",
          "9300/tcp": "32895"
        }
      }
    }
  ]
}
```


4. 这里提供了 Elasticsearch 监听的 9200 和 9300 映射的端口，可与 Elasticsearch 一起使用。

Memcached

概述：

Memcached 是一个高性能的分布式内存对象缓存系统，用于动态 Web 应用以减轻数据库负载。它通过在内存中缓存数据和对象来减少读取数据库的次数，从而提高动态、数据库驱动网站的速度。Memcached 基于一个存储键/值对的 hashmap。其守护进程（daemon ）是用 C 写的，但是客户端可以用任何语言来编写，并通过 memcached 协议与守护进程通信。

功能：

memcached 是一套分布式的快取系统，当初是 Danga Interactive 为了 LiveJournal 所发展的，但被许多软件(如 MediaWiki)所使用。这是一套开放源代码软件，以 BSD license 授权协议发布。

memcached 缺乏认证以及安全管制，这代表应该将 memcached 服务器放置在防火墙后。

memcached 的 API 使用 32 位元的循环冗余校验（CRC-32）计算键值后，将资料分散在不同的机器上。当表格满了以后，接下来新增的资料会以 LRU 机制替换掉。由于 memcached 通常只是当作快取系统使用，所以使用 memcached 的应用程式在写回较慢的系统时（像是后端的数据库）需要额外的程式码更新 memcached 内的资料

memcached 是以 LiveJournal 旗下 Danga Interactive 公司的 Brad Fitzpatric 为首开发的一款软件。已成为 mixi、hatena、Facebook、Vox、LiveJournal 等众多服务中提高 Web 应用扩展性的重要因素。许多 Web 应用都将数据保存到 RDBMS 中，应用服务

器从中读取数据并在浏览器中显示。但随着数据量的增大、访问的集中，就会出现 RDBMS 的负担加重、数据库响应恶化、网站显示延迟等重大影响。

这时就该 memcached 大显身手了。memcached 是高性能的分布式内存缓存服务器。一般的使用目的是，通过缓存数据库查询结果，减少数据库访问次数，以提高动态 Web 应用的速度、提高可扩展性。

Memcached 的守护进程（daemon ）是用 C 写的，但是客户端可以用任何语言来编写，并通过 memcached 协议与守护进程通信。但是它并不提供冗余（例如，复制其 hashmap 条目）；当某个服务器 S 停止运行或崩溃了，所有存放在 S 上的键/值对都将丢失。

Memcached 由 Danga Interactive 开发 ,其最新版本发布于 2010 年 ,作者为 Anatoly Vorobey 和 Brad Fitzpatrick。用于提升 LiveJournal . com 访问速度的。LJ 每秒动态页面访问量几千次，用户 700 万。Memcached 将数据库负载大幅度降低，更好的分配资源，更快速访问。

事件处理

libevent 是个程序库，它将 Linux 的 epoll、BSD 类操作系统的 kqueue 等事件处理功能封装成统一的接口。即使对服务器的连接数增加，也能发挥 O(1)的性能。memcached 使用这个 libevent 库，因此能在 Linux、BSD、Solaris 等操作系统上发挥其高性能。关于事件处理这里就不再详细介绍，可以参考 Dan Kegel 的 The C10K Problem。

存储方式

为了提高性能，memcached 中保存的数据都存储在 memcached 内置的内存存储空

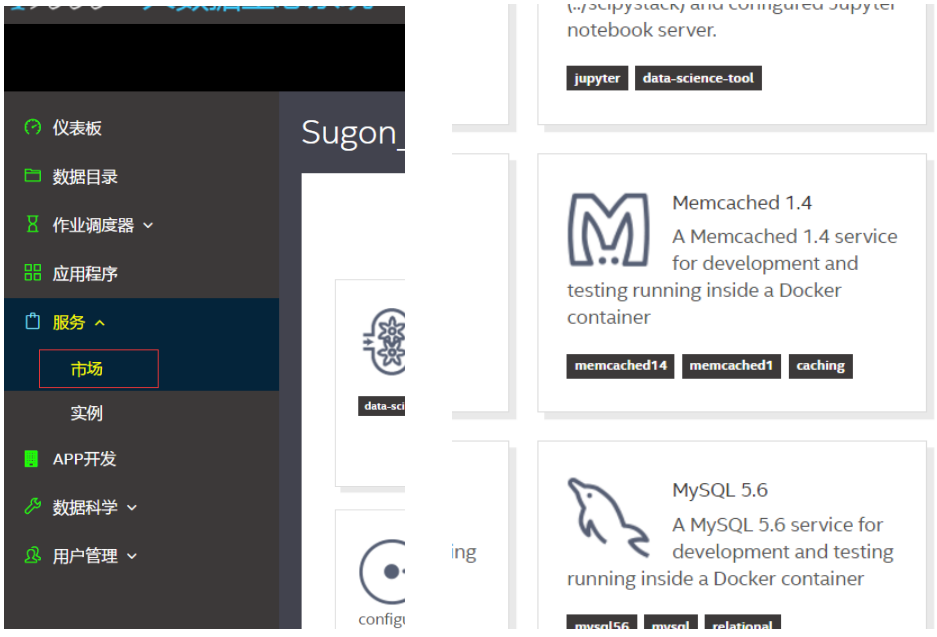
间中。由于数据仅存在于内存中，因此重启 memcached、重启操作系统会导致全部数据消失。另外，内容容量达到指定值之后，就基于 LRU(Least Recently Used)算法自动删除不使用的缓存。memcached 本身是为缓存而设计的服务器，因此并没有过多考虑数据的永久性问题。

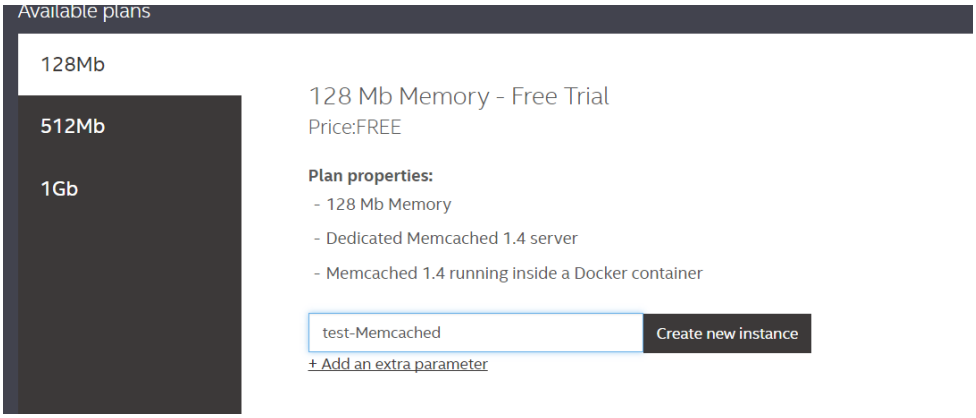
通信分布式

memcached 尽管是“分布式”缓存服务器，但服务器端并没有分布式功能。各个 memcached 不会互相通信以共享信息。那么，怎样进行分布式呢？这完全取决于客户端的实现。本文也将介绍 memcached 的分布式。

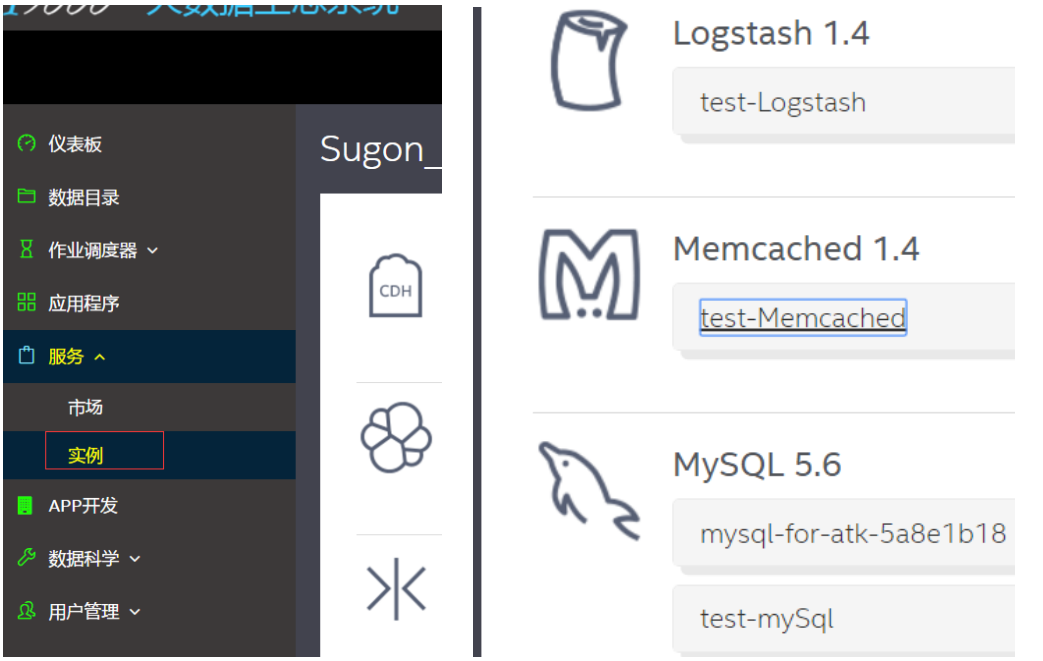
使用方法：

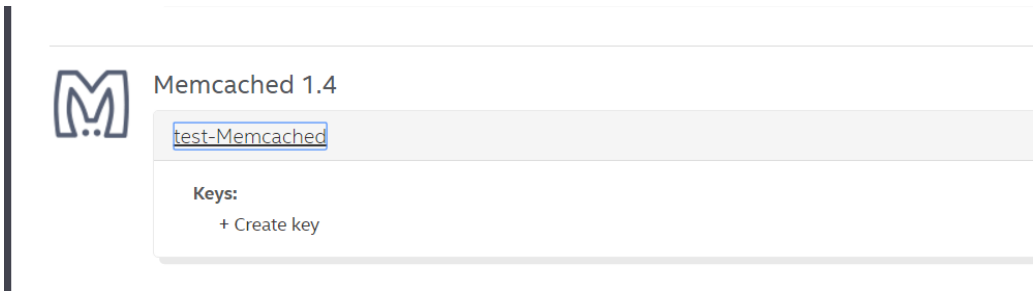
1. 在 i9000 平台上选择 服务>市场，找到 Memcached，点击 Memcached，创建一个新的实例。



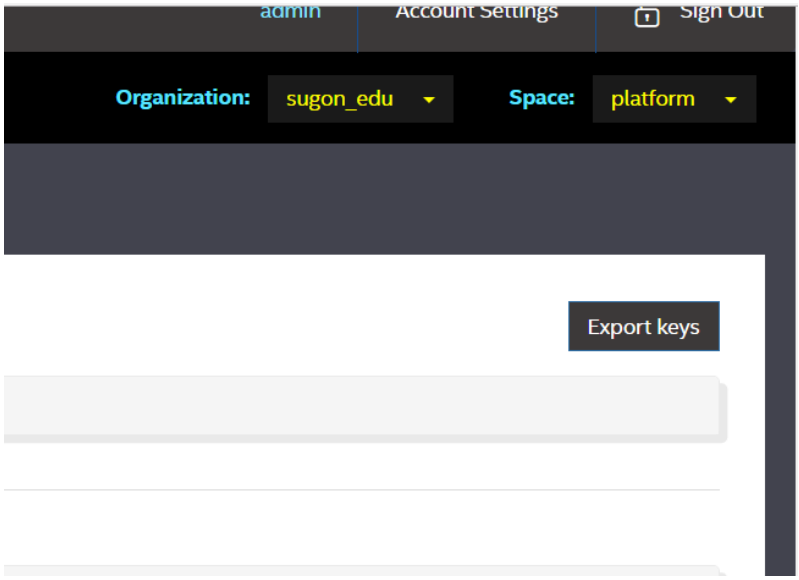


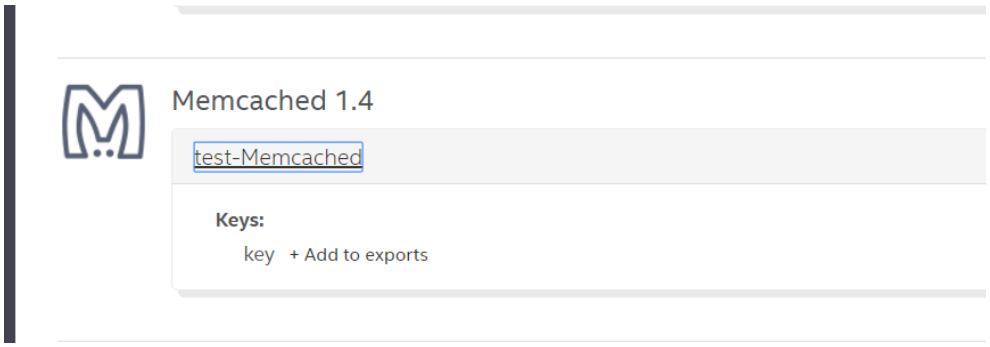
2. 在 服务>实例 里面找到 Memcached，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Memcached 的连接信息。

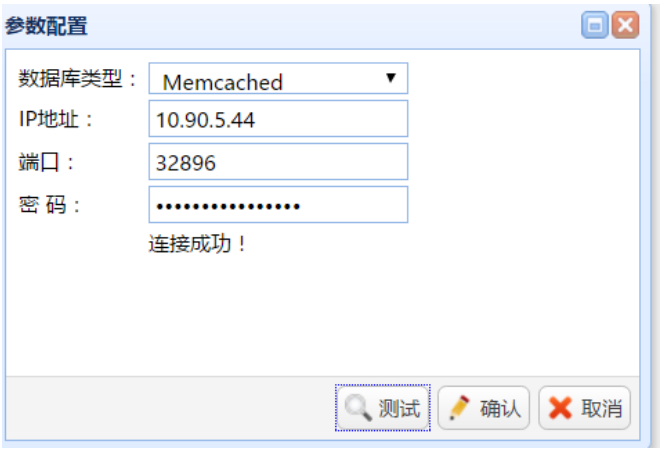




exported Keys:


```
{
  "memcached14": [
    {
      "label": "memcached14",
      "name": "test-Memcached",
      "plan": "128Mb",
      "tags": [
        "memcached14",
        "memcached1",
        "caching "
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "11211/tcp": "32896"
        },
        "port": "32896",
        "username": "4qtrcuip3sbczojg",
        "password": "rg5goj2nhr1yblbm"
      }
    }
  ]
}
```


4. 在可视化客户端（如 TreeNMS）里，可以通过 hostname，port 以及 password 连接到刚刚创建的 Memcached 数据库。


参数配置
数据库类型：
Memcached
IP地址：
10.90.5.44
端口：
32896
密 码：
连接成功！
测试
确认
取消

MongoDB 2.6

简介：

MongoDB 是一个基于分布式文件存储的数据库。由 C++语言编写。旨在为 WEB 应用提供可扩展的高性能数据存储解决方案。

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富，最像关系数据库的。他支持的数据结构非常松散，是类似 json 的 bson 格式，因此可以存储比较复杂的数据类型。Mongo 最大的特点是他支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

特点：

它的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

*面向集合存储，易存储对象类型的数据。

*模式自由。

*支持动态查询。

*支持完全索引，包含内部对象。

*支持查询。

*支持复制和故障恢复。

*使用高效的二进制数据存储，包括大型对象（如视频等）。

*自动处理碎片，以支持云计算层次的扩展性。

*支持 RUBY，PYTHON，JAVA，C++，PHP，C#等多种语言。

*文件存储格式为 BSON（一种 JSON 的扩展）。

*可通过网络访问。

实用原理：

所谓“面向集合”（Collection-Oriented），意思是数据被分组存储在数据集中，被称为一个集合（Collection)。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似关系型数据库（RDBMS）里的表（table），不同的是它不需要定义任何模式（schema)。Nytro MegaRAID 技术中的闪存高速缓存算法，能够快速识别数据库内大数据集中的热数据，提供一致的性能改进。

模式自由（schema-free)，意味着对于存储在 mongodb 数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，你完全可以把不同结构的文件存储在同一个数据库里。

存储在集合中的文档，被存储为键-值对的形式。键用于唯一标识一个文档，为字符串类型，而值则可以是各种复杂的文件类型。我们称这种存储形式为 BSON。

MongoDB 已经在多个站点部署，其主要场景如下：

1）网站实时数据处理。它非常适合实时的插入、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

2）缓存。由于性能很高，它适合作为信息基础设施的缓存层。在系统重启之后，由它搭建的持久化缓存层可以避免下层的数据源过载。

3）高伸缩性的场景。非常适合由数十或数百台服务器组成的数据库，它的路线图中已经包含对 MapReduce 引擎的内置支持。

不适用的场景如下：

1）要求高度事务性的系统。

2）传统的商业智能应用。

3）复杂的跨文档（表）级联查询。

系统介绍：

分布式文件系统（Distributed File System）是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。分布式文件系统的设计基于客户机/服务器模式。一个典型的网络可能包括多个供多用户访问的服务器。另外，对等特性允许一些系统扮演客户机和服务器的双重角色。

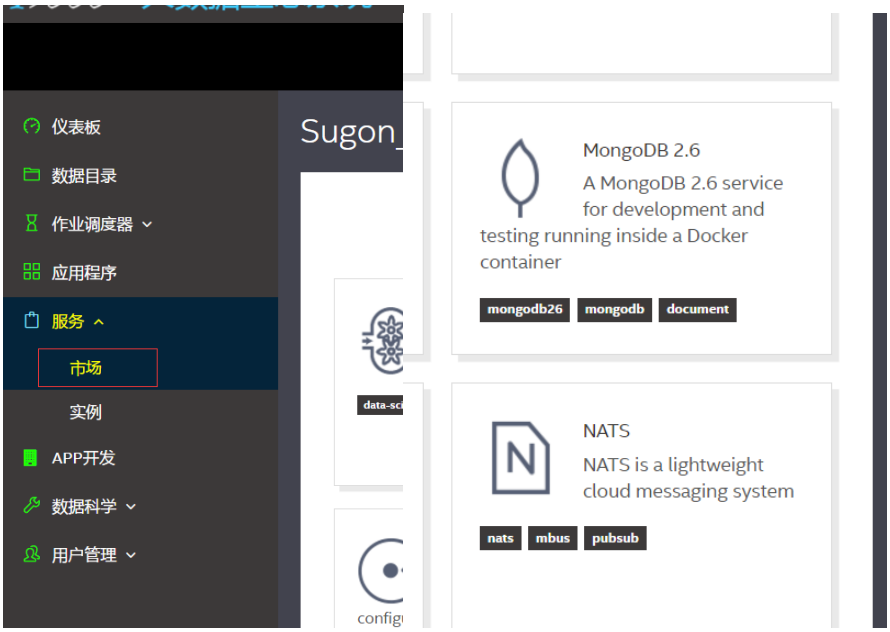
HBase 是一个分布式的、面向列的开源数据库，该技术来源于 Fay Chang 所撰写的 Google 论文“Bigtable：一个结构化数据的分布式存储系统。

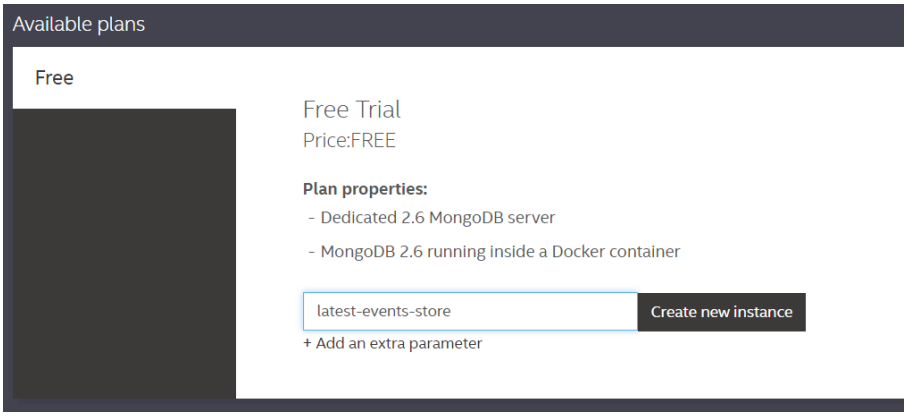
Yonghong Data Mart 是基于自有技术研发的一款数据存储、数据处理的软件。Yonghong Data Mart 的分布式文件存储系统 (ZDFS)是在 Hadoop HDFS 基础上进行的改造和扩展，将服务器集群内所有节点上存储的文件统一管理和存储。

使用方法：

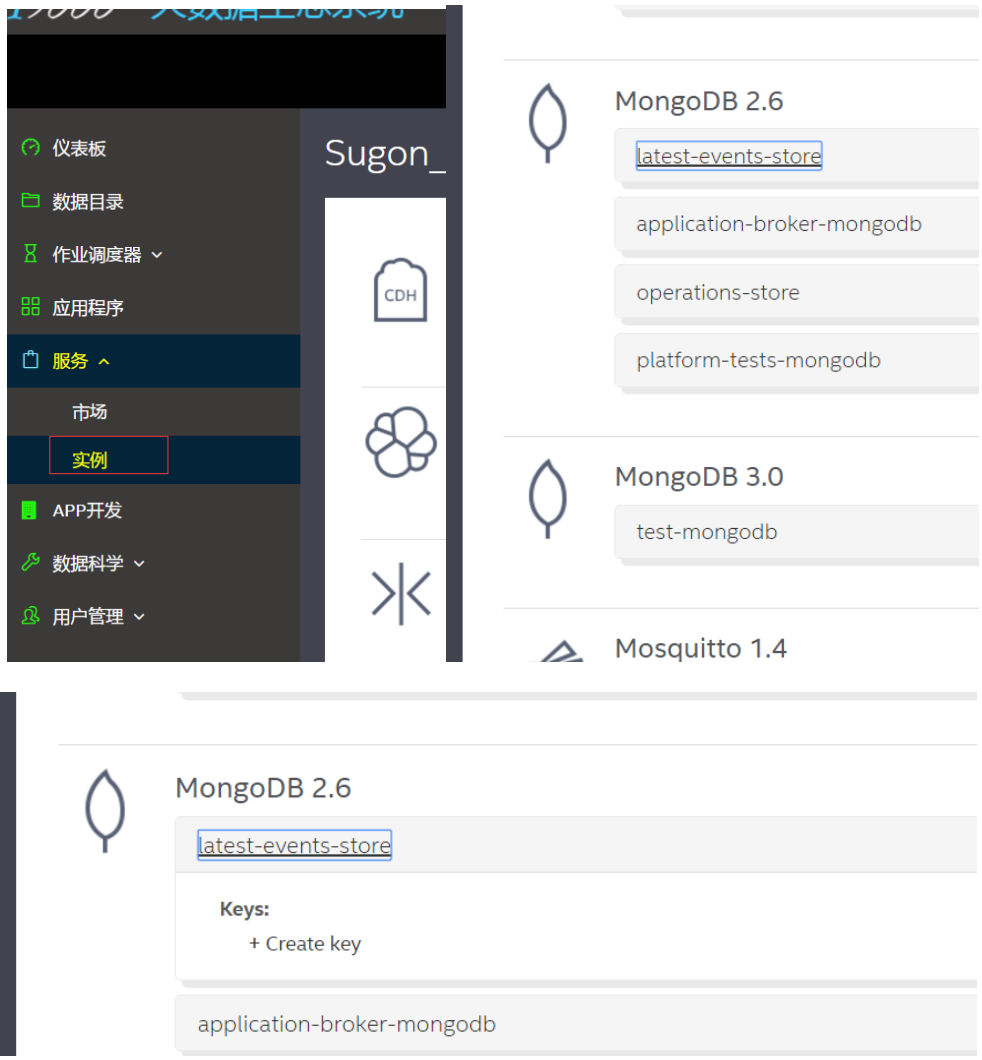
1. 在 i9000 平台上选择 服务>市场，找到 MongoDB 2.6，点击 MongoDB 2.6，创建一

个新的实例。

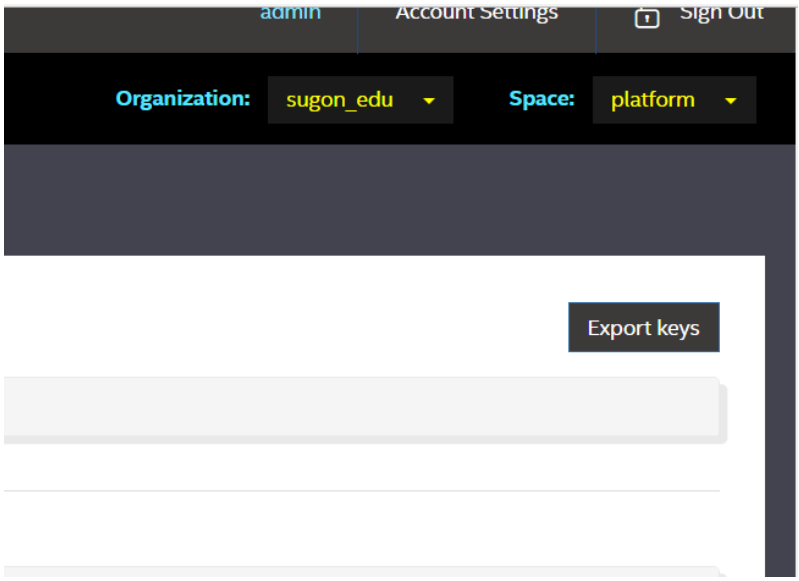


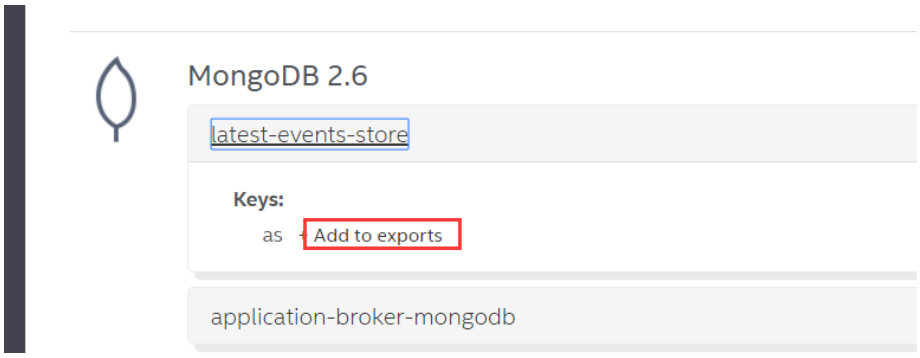


2. 在 服务>实例 里面找到 MongoDB 2.6，找到刚刚创建的实例，点击，创建新的 key。



3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 MongoDB 2.6 的连接信息。





4. 在 MongoDB 客户端里，可以通过 hostname，port 以及 username，password 连接到刚刚创建的 MongoDB 数据库。

MongoDB 3.0

相比于 **MongoDB 2.6** 新版的特点介绍：

1. 引入了插件式的存储引擎架构，允许第三方根据实际项目的需要开发存储引擎，类似于 MySql 中的分层存储引擎架构。随同这种架构发布的新的存储引擎为 WiredTiger，老的存储引擎更名为 MMAPv1，也是目前默认的存储引擎。

WiredTiger 存储引擎主要特点有：

（1）目前只支持 64 位的 MongoDB。

（2）支持文档级别的锁，相当于 MySql 中的行级别锁，多个客户端能够同时修改同一个集合中的多个文档，相比以前版本，WiredTiger 存储引擎带来了更细粒度的锁，因此 MongoDB 的并发性能得到了大大的提高。

（3）支持集合和索引的压缩存储，这样能减少存储的消耗而只需要较少的 CPU 开支。

2. 不同存储引擎对应的数据文件不能兼容，也就是说以前版本对应的数据文件不能直接被 WiredTiger 存储引擎所支持，如果要升级到 3.0 的 WiredTiger 存储引擎，必须先将版本升级到 2.6，然后再升级到 3.0，下面介绍一个单实例升级步骤：

Step1：下载最新的 3.0 版本的 mongod 二进制文件，取代 2.6 版本的二进制。

Step2：启动 3.0 版本的 mongod 进程，确保使用的是默认存储引擎 MMAPv1。

Step3：利用 mongodump 导出数据文件。

Step4：创建新的数据目录为 WiredTiger 存储引擎。

Step5：重启 mongod 实例用 WiredTiger 存储引擎选项，如下：

mongod –storageEngine wiredTiger –dbpath ‘Step4 创建的新目录’

Step6：使用 mongorestore 恢复数据文件。

3. 默认的存储引擎 MMAPv1 在 3.0 版本中也有所改进，支持了集合级别的锁，相当于 MySql 中的表级别锁，以前版本都是数据库级别和全局实例级别的锁，锁的粒度有所降低。

4．复制集的也发生了一点改变，在 3.0 版本中，首先允许的成员最大数量增加了，可以达到 50 个；其次复制集中 primary 节点关闭时，复制集的行为也有所变化，在 3.0 版本中，复制集会终止耗时较长的操作，例如索引的构造，map-reduce 作业等，复制集会等到有新的 priamry 节点选出后原来的 primary 节点才会关闭，而以前的版本只是简单的等待 10 秒，不管是否选出了新的 primary 节点，同时现在也可以指定参数 secondaryCatchUpPeriodSecs 值，明确指定等待多少秒。

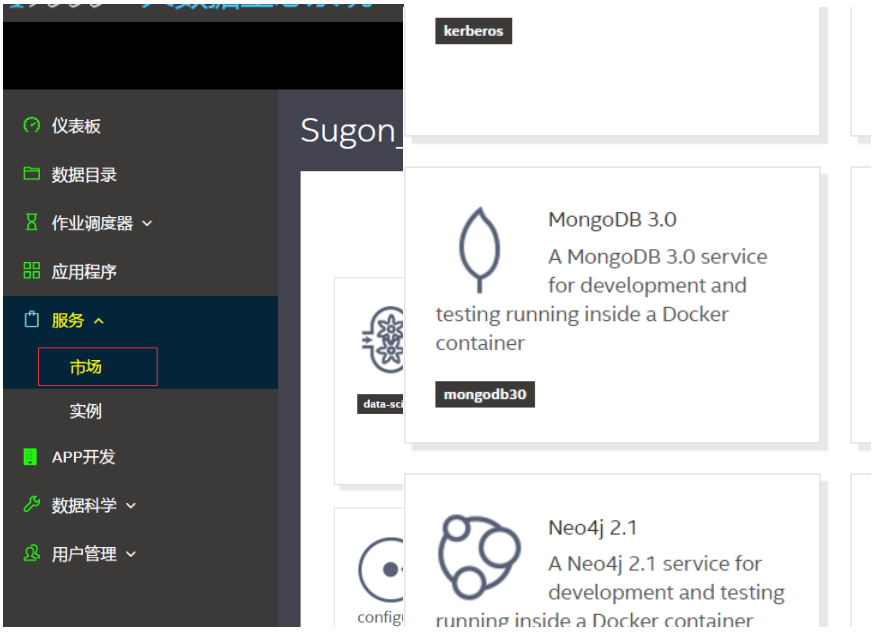
5. 工具的改变，mongodump，mongorestore，mongoexport，mongoimport，mongofiles 以及 mongooplog 工具必须连接到正在运行的 mongod 实例上进行操作，不能像以前版本那样直接通过—dbpath 选项操作数据文件。

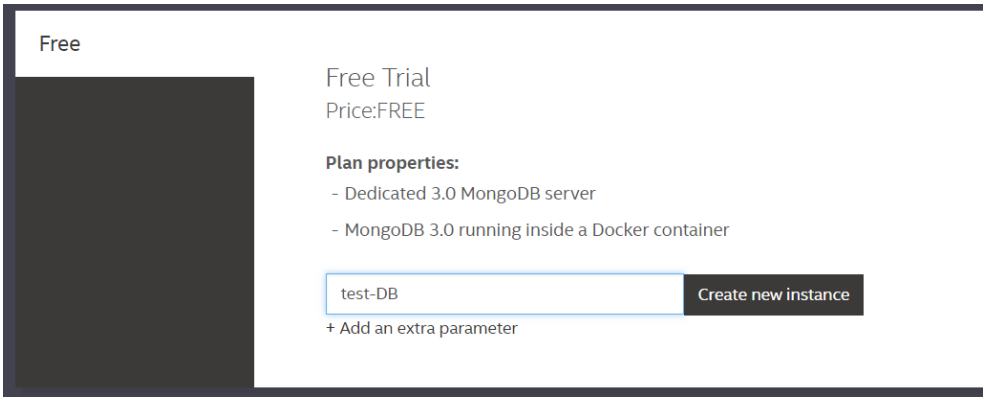
6. 本地连接权限的改变，本地连接只能创建第一个用户在 admin 数据库中，不像以前版本，本地连接没有权限限制，能在实例上做任何操作。同时 db.addUser()命令被废弃了，用 db.createUser()和 db.updateUser()来代替。

使用方法：

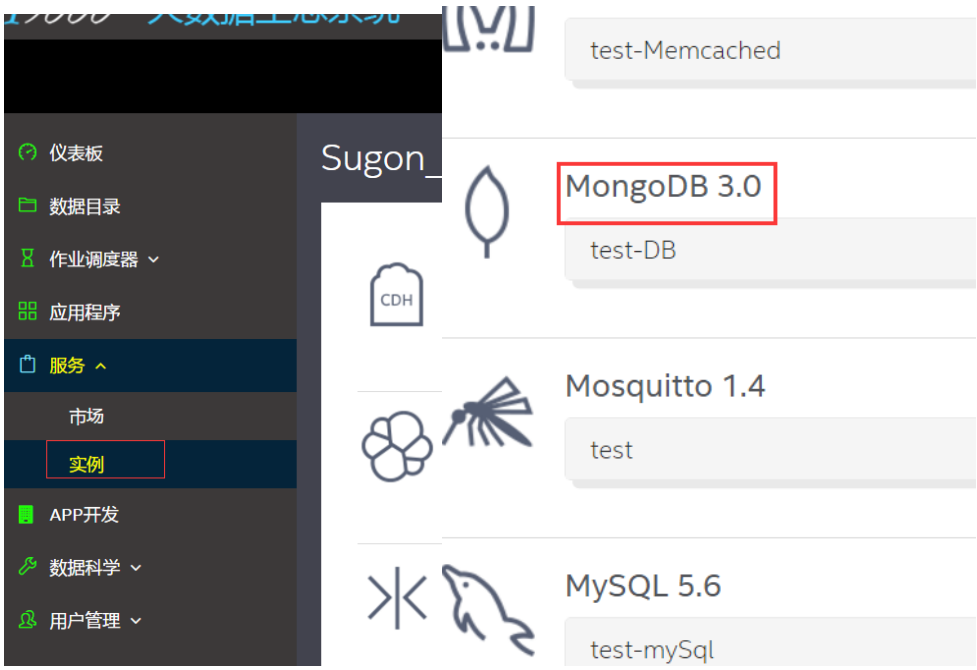
1. 在 i9000 平台上选择 服务>市场，找到 MongoDB 3.0，点击 MongoDB 3.0，创建一

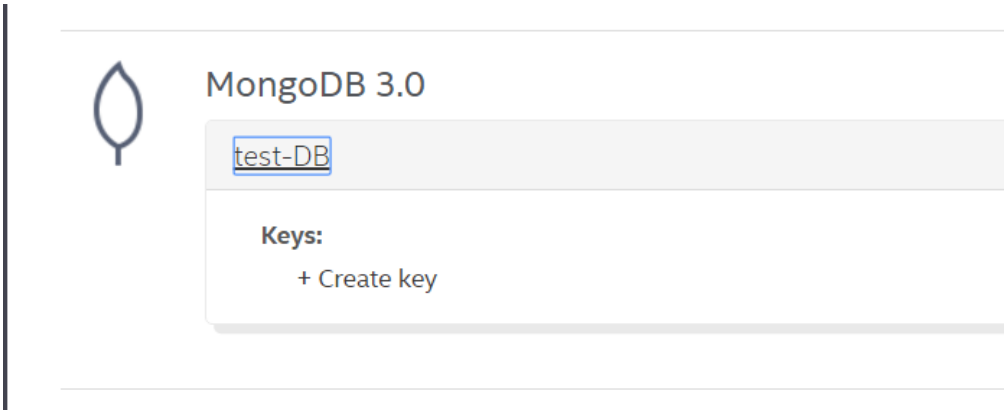
个新的实例。



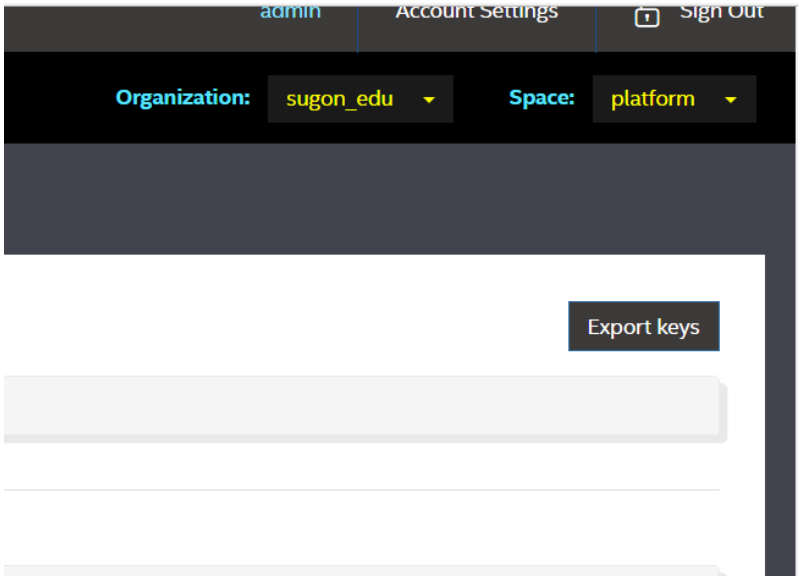


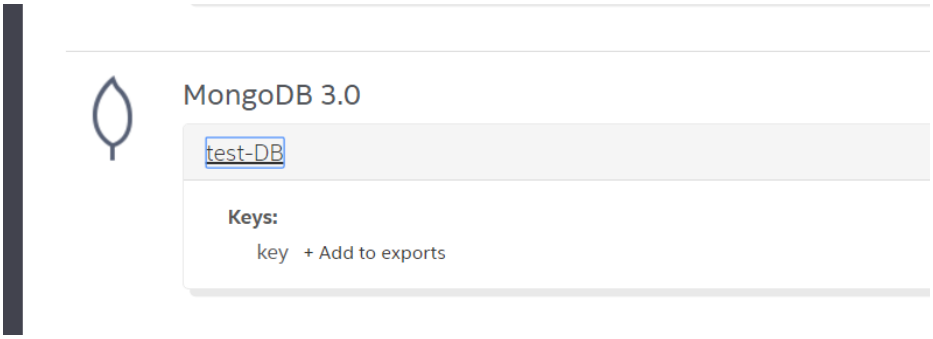
2. 在 服务>实例 里面找到 MongoDB 3.0，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 MongoDB 3.0 的连接信息。






```
{
  "mongodb30": [
    {
      "label": "mongodb30",
      "name": "test-DB",
      "plan": "free",
      "tags": [
        "mongodb30"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "27017/tcp": "32938",
          "28017/tcp": "32939"
        },
        "port": "32938",
        "username": "x08nzbahjfuzr0gw",
        "password": "zzfphjavxuf2kfqs",
        "dbname": "sjfphq1cgx2gpubs",
        "uri": "mongodb://x08nzbahjfuzr0gw:zzfphjavxuf2kfqs@10.0.4.4:32938/sjfphq1cgx2gpubs"
      }
    }
  ]
}
```


4. 在 MongoDB 可视化工具里里，可以通过 hostname，port 以及 username，password 连接到刚刚创建的 MongoDB 数据库。

Mosquitto 1.4

Mosquitto 1.4 是一款实现了消息推送协议 Mosquitto 的开源消息代理软件，提供轻量级的，支持可发布/可订阅的的消息推送模式，使设备对设备之间的短消息通信变得简单，比如现在应用广泛的低功耗传感器，手机、嵌入式计算机、微型控制器等移动设备。一个典型的应用案例就是 Andy Stanford-ClarkMosquitto（Mosquitto 协议创始人之一）在家中实现的远程监控和自动化。并在 OggCamp 的演讲上，对 Mosquitto 协议进行详细阐述。

Mosquitto 采用代理的发布/订阅模式实现了发布者和订阅者的解耦(decouple)，因此，在 Mosquitto 协议中有三种角色：代理服务器、发布者客户端以及订阅者客户端，其中发布者和订阅者互不干扰，也就是说发布者和订阅者互不知道对方的存在，它们只知道代理服务器，代理服务器负责将来自发布者的消息进行存储处理并将这些消息发送到正确的订阅者中去。这种解耦体现在以下 3 个方面上：

·空间解耦：发布者和订阅者不必知道对方的存在，例如对方的 IP 地址或者端口；

·时间解耦：发布者和订阅者不必同时建立连接；

·同步解耦：发布者和订阅者在发布消息或接收消息的时候不需要同步；

MOSQUITTO 的消息类型：

1.CONNECT 控制报文用于客户端请求与服务器建立连接，应用层的连接而不是 TCP/IP 连接，CONNECT 控制报文的发送在 TCP/IP 建立连接后；

2.CONNACK 控制报文用于服务器向请求连接的客户端回发连接确认；

3.PUBLISH 控制报文用于发布指定主题名的应用信息；

4.PUBACK/PUBREC/PUBREL/PUBCOMP 控制报文用于针对不同服务质量的应用信息的回应；

5.SUBSCRIBE 控制报文用于订阅者向服务器发送一个主题过滤器列表，用于表示客户端想要订阅的主题；

6.SUBACK 控制报文用于 SUBSCRIBE 控制报文的响应；

7.UNSUBSCRIBE 控制报文用于向服务器发送一个主题过滤器列表，用于表示客户端想要取消订阅的主题；

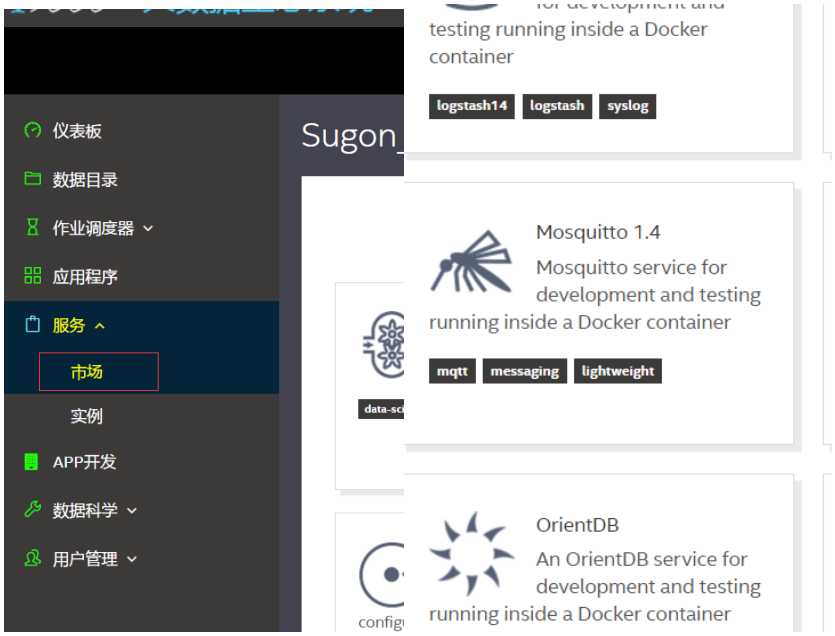
8.UNSUBACK 控制报文用于 UNSUBSCRIBE 控制报文的响应；

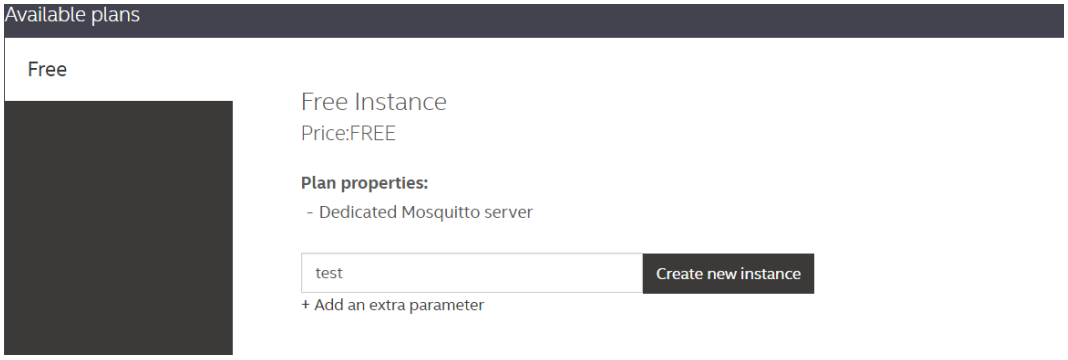
9.PINGREQ/PINGRESP 控制报文作为客户端和服务器间的心跳包；

10.DISCONNECT 控制报文用于客户端在断开前告诉服务器其将断开连接；

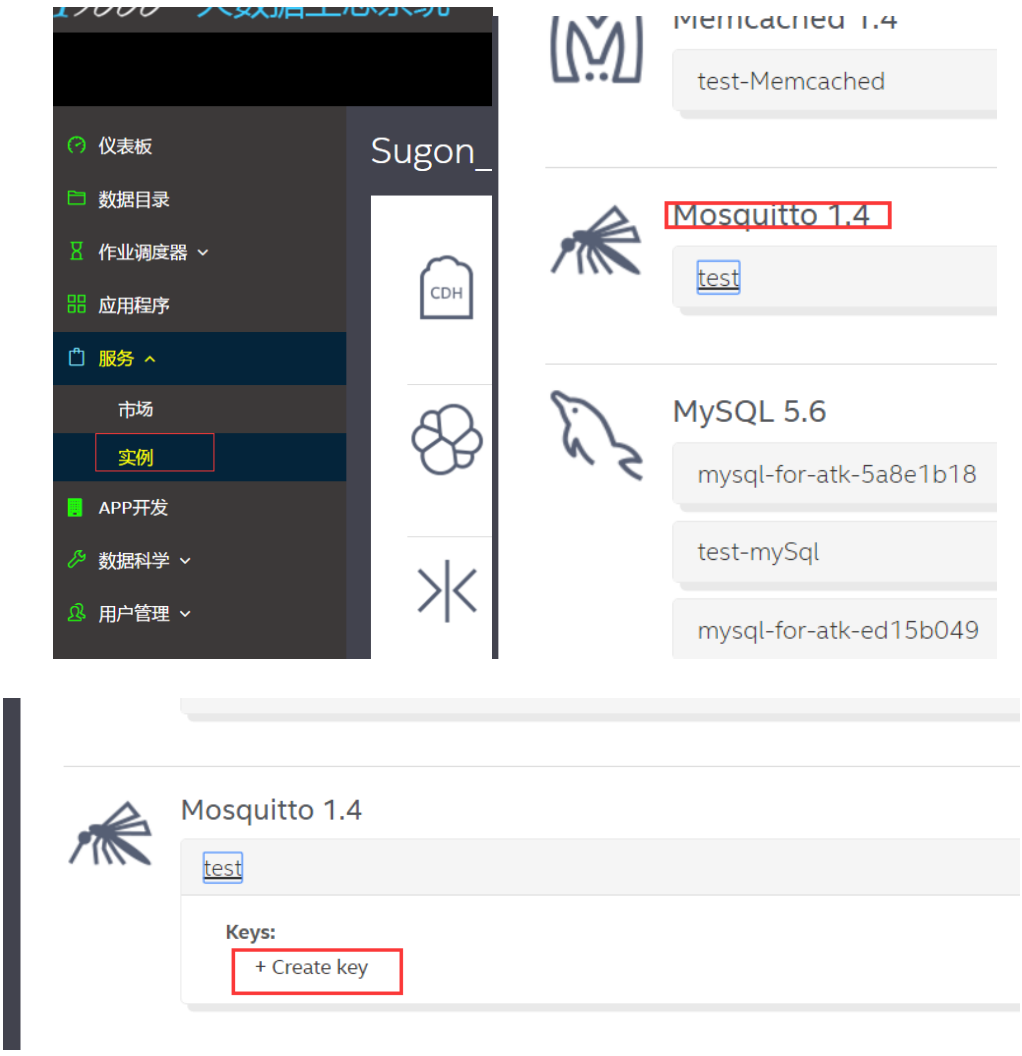
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 Mosquitto，点击 Mosquitto，创建一个新的实例。

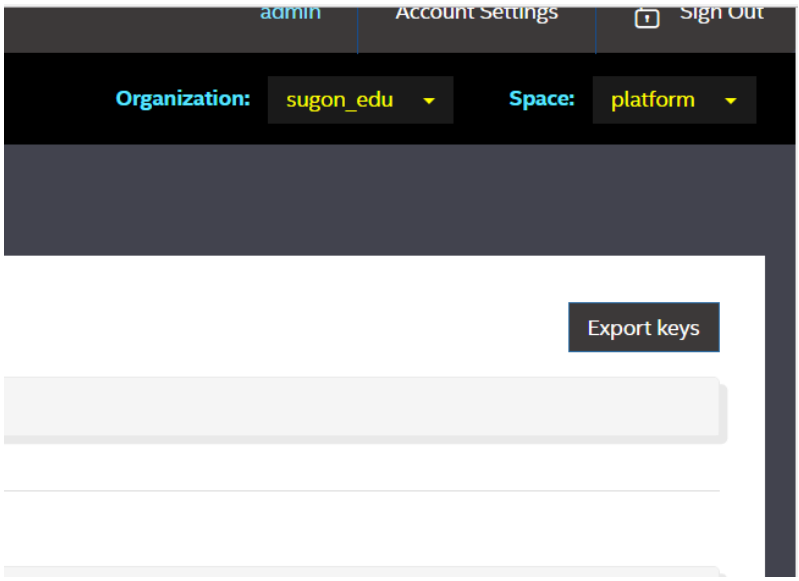


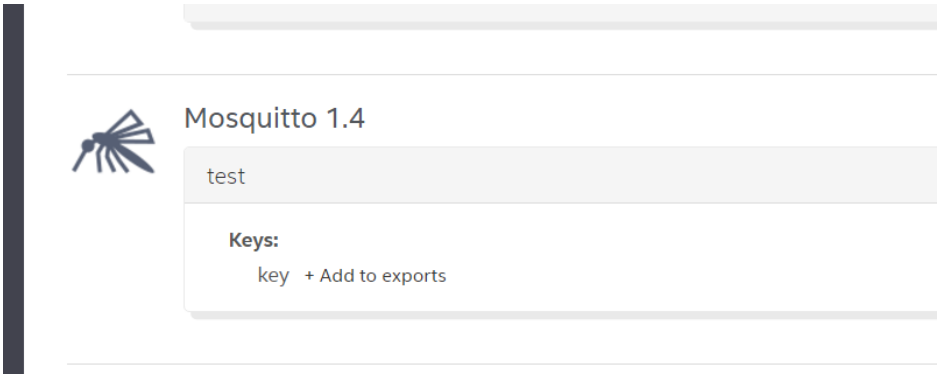


2. 在 服务>实例 里面找到 Mosquitto，找到刚刚创建的实例，点击，创建新的 key。



3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Mosquitto 的连接信息。





exported Keys:


```
{
  "mosquitto14": [
    {
      "label": "mosquitto14",
      "name": "test",
      "plan": "free",
      "tags": [
        "mqtt",
        "messaging",
        "lightweight"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "1883/tcp": "32898"
        },
        "port": "32898",
        "username": "2sudtstkvq1p8rch",
        "password": "weo1vobufktsynix"
      }
    }
  ]
}
```


4. 这里可以看到 Mosquitto 的链接信息，hostname 说明所连接到的域名，port 是映射

的端口，用户名密码用于代理认证。可在其他实例或自己发布应用是绑定使用。

MySQL 5.6

概述：

MySQL 是一个关系型数据库管理系统，由瑞典 MySQL AB 公司开发，目前属于 Oracle 旗下产品。MySQL 最流行的关系型数据库管理系统，在 WEB 应用方面 MySQL 是最好的 RDBMS (Relational Database Management System，关系数据库管理系统)应用软件之一。

MySQL 是一种关联数据库管理系统，关联数据库将数据保存在不同的表中，而不是将所有数据放在一个大仓库内，这样就增加了速度并提高了灵活性。

MySQL 所使用的 SQL 语言是用于访问数据库的最常用标准化语言。MySQL 软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发都选择 MySQL 作为网站数据库。

由于其社区版的性能卓越，搭配 PHP 和 Apache 可组成良好的开发环境。

应用环境：

与其他的大型数据库例如 Oracle、DB2、SQL Server 等相比，MySQL 自有它的不足之处，但是这丝毫也没有减少它受欢迎的程度。对于一般的个人使用者和中小型企业来说，MySQL 提供的功能已经绰绰有余，而且由于 MySQL 是开放源码软件，因此可以大大降低总体拥有成本。

Linux 作为操作系统，Apache 或 Nginx 作为 Web 服务器，MySQL 作为数据库，PHP/Perl/Python 作为服务器端脚本解释器。由于这四个软件都是免费或开放源码软件（FLOSS)，因此使用这种方式不用花一分钱（除开人工成本）就可以建立起一个稳定、免

费的网站系统，被业界称为“LAMP”或“LNMP”组合

存储引擎：

MMySQL 之前的默认数据库引擎，最为常用。拥有较高的插入，查询速度，但不支持事务

InnoDB 事务型数据库的首选引擎，支持 ACID 事务，支持行级锁定, MySQL 5.5 起成为默认数据库引擎

BDB 源 自 Berkeley DB ,事务型数据库的另一种选择 ,支持 Commit 和 Rollback 等其他事务特性

Memory 所有数据置于内存的存储引擎，拥有极高的插入，更新和查询效率。但是会占用和数据量成正比的内存空间。并且其内容会在 MySQL 重新启动时丢失

Merge 将一定数量的 MyISAM 表联合而成一个整体，在超大规模数据存储时很有用

Archive 非常适合存储大量的独立的，作为历史记录的数据。因为它们不经常被读取。Archive 拥有高效的插入速度，但其对查询的支持相对较差

Federated 将不同的 MySQL 服务器联合起来，逻辑上组成一个完整的数据库。非常适合分布式应用

Cluster/NDB 高冗余的存储引擎，用多台数据机器联合提供服务以提高整体性能和安全性。适合数据量大，安全和性能要求高的应用

CSV： 逻辑上由逗号分割数据的存储引擎。它会在数据库子目录里为每个数据表创建一个 .csv 文件。这是一种普通文本文件，每个数据行占用一个文本行。CSV 存储引擎不

支持索引。

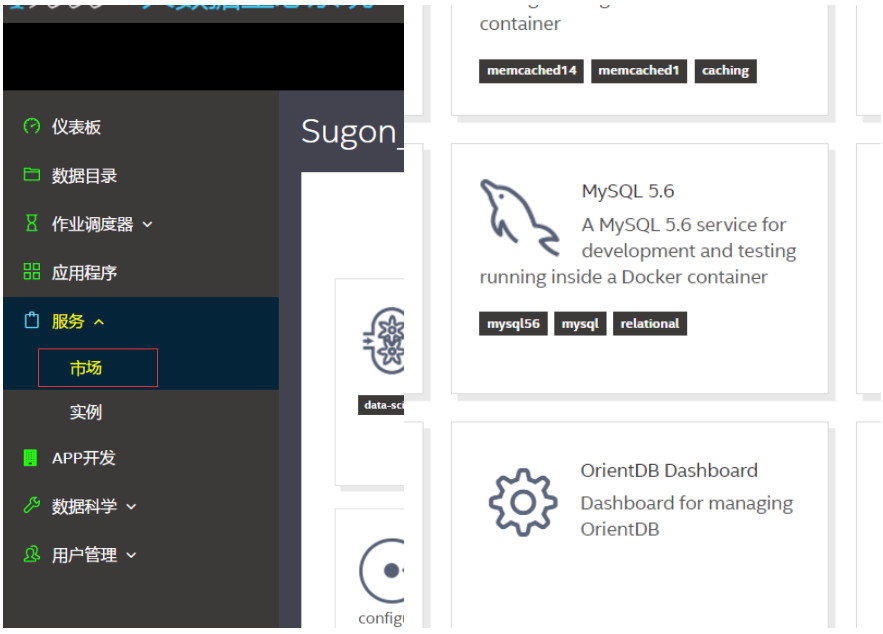
BlackHole：黑洞引擎，写入的任何数据都会消失，一般用于记录 binlog 做复制的中继

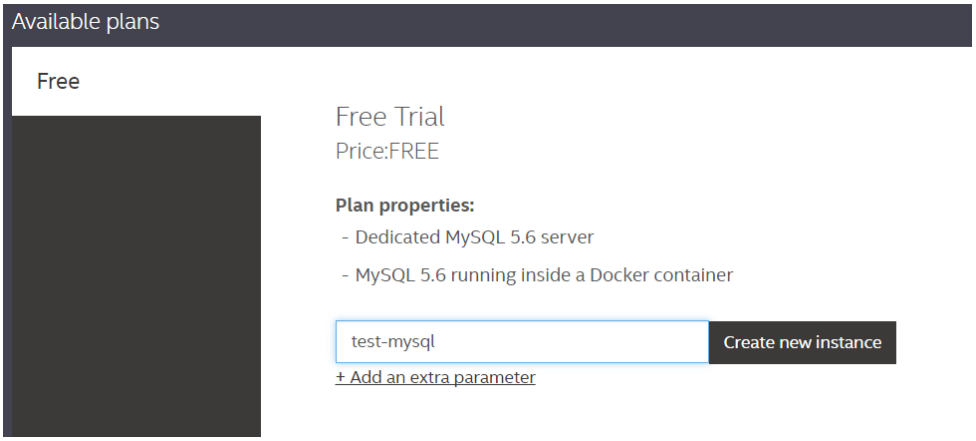
EXAMPLE 存储引擎是一个不做任何事情的存根引擎。它的目的是作为 MySQL 源代码中的一个例子，用来演示如何开始编写一个新存储引擎。同样，它的主要兴趣是对开发者。EXAMPLE 存储引擎不支持编索引。

另外，MySQL 的存储引擎接口定义良好。有兴趣的开发者可以通过阅读文档编写自己的存储引擎。

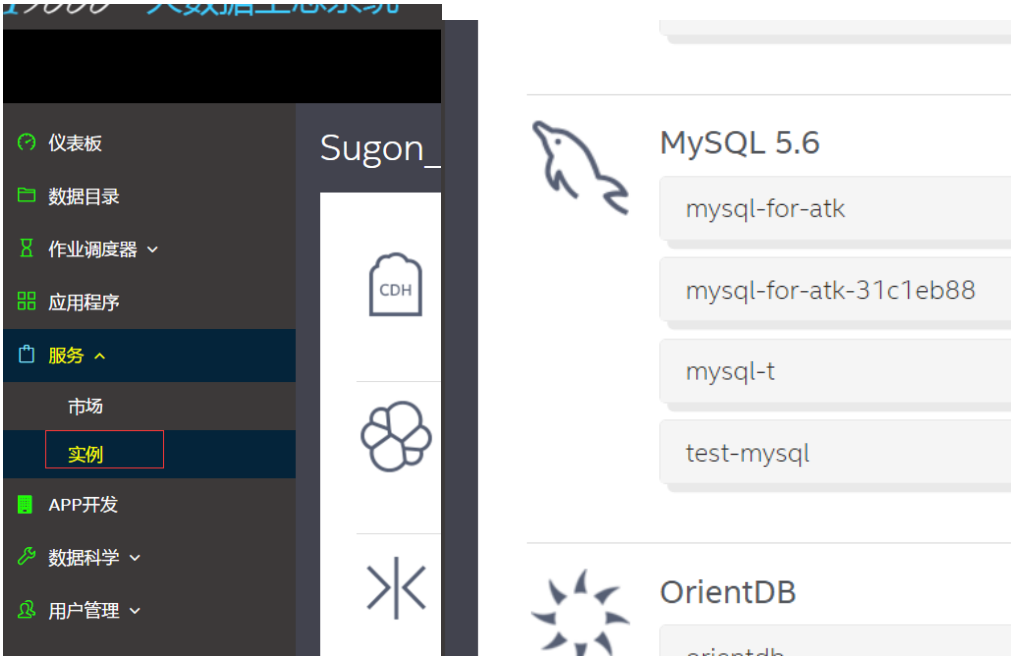
使用方法：

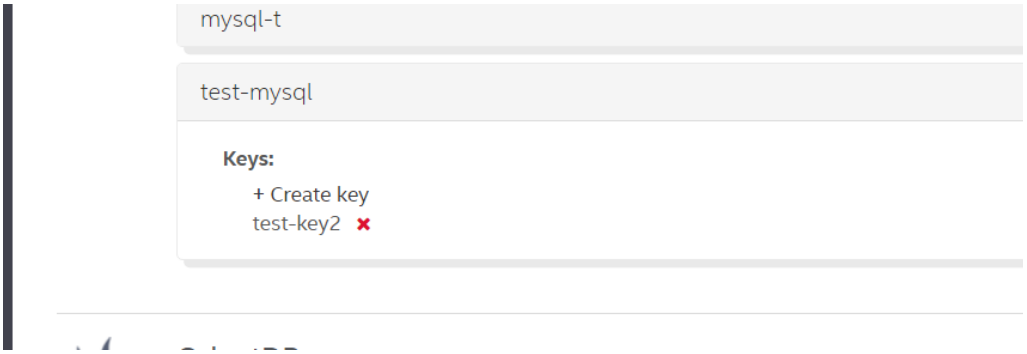
1. 在 i9000 平台上选择 服务>市场，找到 MySQL 5.6，点击 MySQL 5.6，创建一个新的实例。



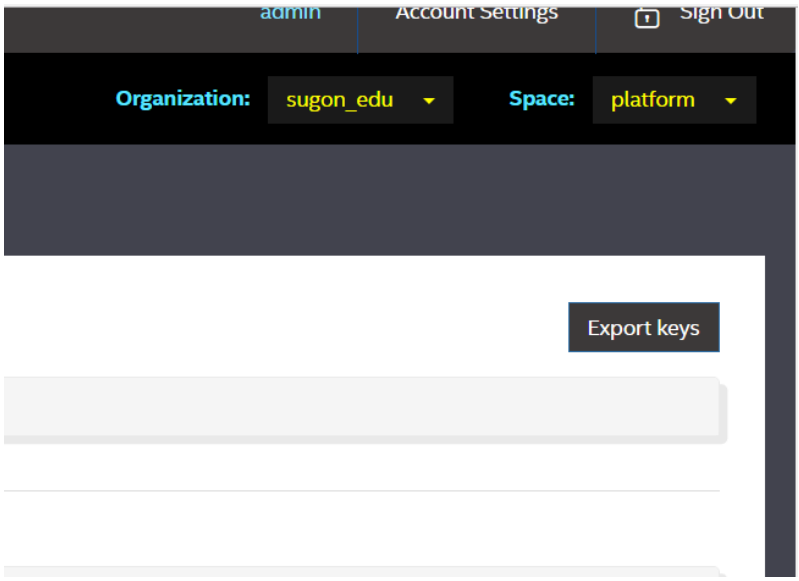


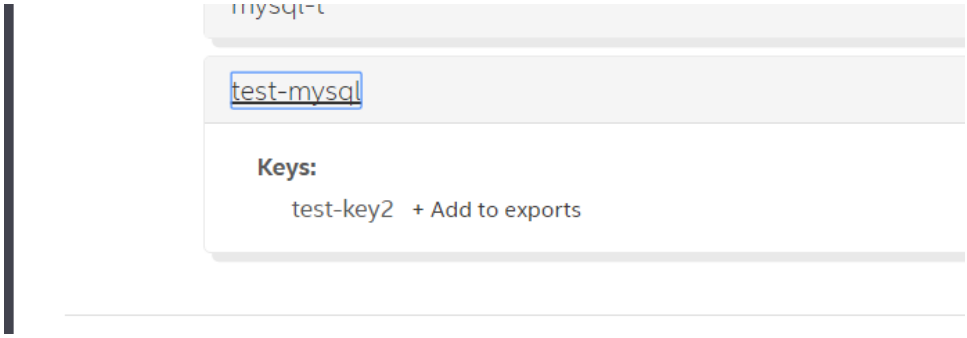
2. 在 服务>实例 里面找到 MySQL 5.6，找到刚刚创建的实例，点击，创建新的 key。





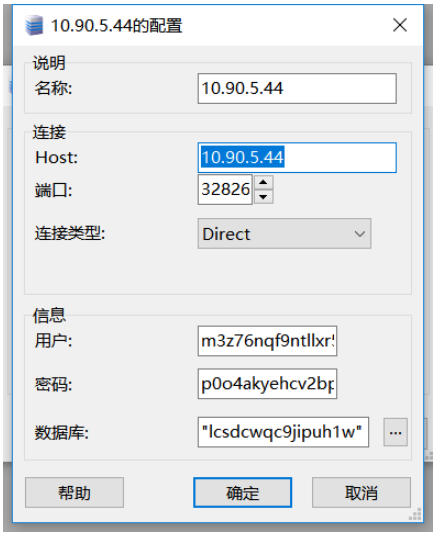
3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 MySQL 5.6 的连接信息。

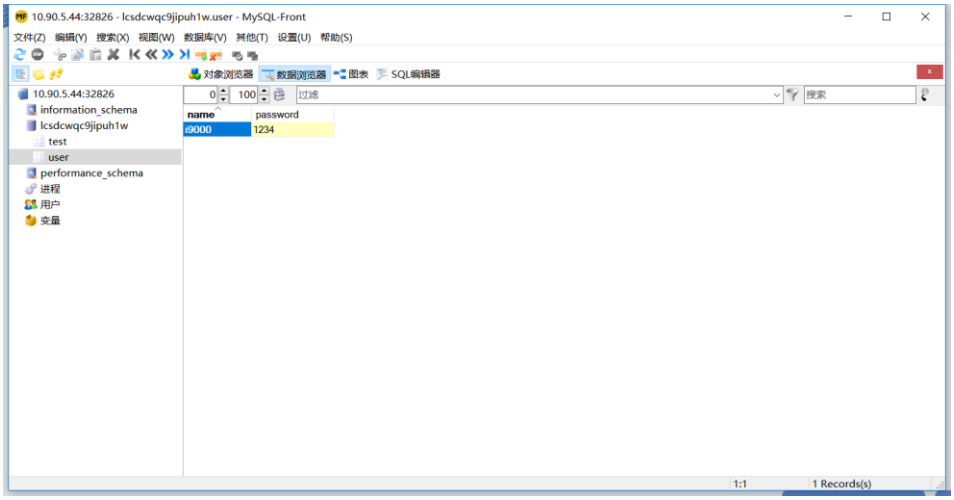





```
"mysql56": [
  {
    "label": "mysql56",
    "name": "test-mysql",
    "plan": "free",
    "tags": [
      "mysql56",
      "mysql",
      "relational"
    ],
    "credentials": {
      "hostname": "10.0.4.4",
      "ports": {
        "3306/tcp": "32797"
      },
      "port": "32797",
      "username": "qohfzg2e0ozzssis",
      "password": "r7dxkh2rnzrfwpas",
      "dbname": "wtyjuf4tdjpjtqnm",
      "uri": "mysql://qohfzg2e0ozzssis:r7dxkh2rnzrfwpas@10.0.4.4:32797/wtyjuf4tdjpjtqnm"
    }
  }
]
}
```


4. 在 MySQL 5.6 客户端里，可以通过 hostname，port 以及 username，password 连接到刚刚创建的 MySQL 5.6 数据库。


10.90.5.44的配置
说明
名称:
10.90.5.44
连接
Host:
10.90.5.44
端口:
32826
连接类型:
Direct
信息
用户:
m3z76nqf9ntllxr
密码:
p0o4akyehcv2bp
数据库:
"lcsdcwqc9jipuh1w"
帮助
确定
取消


10.90.5.44:32826 - lcsdcwqc9jipuh1w.user - MySQL-Front
文件(Z) 编辑(Y) 搜索(X) 视图(W) 数据库(V) 其他(T) 设置(U) 帮助(S)
对象浏览器 数据浏览器 图表 SQL编辑器
10.90.5.44:32826
information_schema
lcsdcwqc9jipuh1w
test
user
performance_schema
进程
用户
变量
name
password
i9000
1234
1:1
1 Records(s)

NATS

概述：

NATS 是一个开源的、轻量级的、高性能的分布式消息通信系统，实现了高可伸缩性和优雅的发布/订阅模型。NATS 适合云基础设施的消息通信系统、IoT 设备消息通信和微服务架构。Apcera 团队负责维护 NATS 服务器（Golang 语言开发）和客户端（包括 Python、Ruby、Node.js、Elixir、Java、Nginx、C 和 C#），开源社区也贡献了一些客户端库，包括 Rust、PHP、Lua 等语言的库。

是基于 CloudFoundry（后文统一使用 CF 作为简称）开源系统进行二次开发的，我们利用 CF 开源系统的基本框架，然后整合了各个云服务产品，并且根据需求开发了智能路由、弹性伸缩、智能启动和资源隔离等扩展功能。

CF 开源系统作为一个通用的 PaaS 平台解决方案，很好的满足了大部分基本需求，但是要打造一个高可靠的 PaaS 平台，还需要做很多架构容错和改进。

CF 由很多组件构成，有一些核心组件是必不可少的，还有很多可选组件可以根据需求选择性使用。其中 NATS、Router 和 Cloudontroller 三个组件更是整个 CF 的最关键的组件，其中任何一个组件不可用都会导致整个 PaaS 平台不可用，所以这三个组件的容错显得更加重要。 Router 的容错可以 VIP 实现，Cloudontroller 本身通过 Router 的软域名映射进行容错。NATS 作为整个 CF 各个组件的连接枢纽，一旦不可用所有 CF 组件都出问题，所以 NATS 的重要性不言而喻，但是目前业界普遍使用单实例。虽然 NATS 的稳定性不容置疑，但是不能解决由于网络、服务器硬件故障和操作系统故障导致的不可用，特别是由于服务器不可用导致的故障，恢复成本很高，时间也是很长的。开源的 NATS 组件也有集群版本，但是普遍反映不够稳定，都没有在生产环境使用。

NATS 集群化方案实现：

NATS 集群化实现的难点主要在于怎样保证消息的订阅与发布能够在各个 NATS 节点之间进行同步。下面分别从 NATS 服务器节点启动、订阅消息、发布消息和 NATS 客户端改进等方面说明 NATS 集群化方案的实现。

1. NATS 服务器节点启动

为了保证各个 NATS 服务器节点订阅信息的同步，启动一个 NATS 服务器节点的时候需要判断是否已存在其他 NATS 服务器节点，如果存在那么需要连接其他 NATS 服务器节点进行订阅消息的同步。NATS 服务器节点的各个功能或者服务初始化完毕以后将自己的地址以临时节点的方式注册到 zookeeper 集群中，这样就可以开始对外提供服务了。

2. 消息订阅

集群化版本的 NATS 对于 NATS 客户端发布订阅消息是透明的，即不管 NATS 客户端选择的哪一个 NATS 服务器节点都只需要向这一台进行消息订阅与发布。

NATS 服务器节点收到订阅消息以后，首先加入自己的消息订阅队列，然后广播到其他 NATS 服务器节点，在广播的时候做了一点点优化，就是看这个主题的消息订阅是否已经被广播过了，那么就不需要重复广播，防止订阅信息过多，并且这些都是不需要的垃圾订阅消息。

最后如果某一个客户端取消订阅消息，同样需要广播取消订阅消息。

3. 消息发布

NATS 服务器节点接收到 NATS 客户端发布的消息以后，还是首先根据消息订阅列表进

行转发，和单机 NATS 不同的是，这次转发可能会转发到其他 NATS 服务器节点，只要其他 NATS 服务器节点也有同样的消息主题订阅，那么这种情况就存在一次中间转发的过程，但是也最多只存在一次中间转发过程，所以性能基本上不受什么影响。

4. NATS 客户端改进

NATS 客户端的改进主要是支持从 zookeeper 集群上获取 NATS 服务器的节点地址，并且能够兼容以前老的直接配置某一个 NATS 服务器节点地址。当客户端第一次连接 NATS 服务器节点时，需要从 zookeeper 集群获取所有可用的 NATS 服务器节点地址并且缓存到本地，然后随机选择一个 NATS 服务器节点建立连接并且进行消息订阅与发布。缓存所有 NATS 服务器节点地址主要是防止 zookeeper 不可用的时候并且 NATS 服务器节点有挂掉的情况不影响客户端切换 NATS 服务器节点，在切换的时候需要把 NATS 客户端以前订阅的消息全部重新订阅一次。

其他改进

本次针对 NATS 单点问题进行整个云擎的架构升级，完成升级以后整体运行很稳定。不过除了 NATS 集群化，本次架构升级还做了其他很多改进，本次架构升级主要目的是让京东云擎具有高可靠。下面在简单介绍本次架构升级其他方面的改进：

a) 使用分布式文件系统替换原来的单磁盘存放 droplet，解决了由于用户猛增导致 droplet 存储空间受限的问题；
b) 用户控制台界面 dashboard 的改变；
c) Router 对后端多实例包括 CloudController 的容错；
d) 独立域名绑定支持；

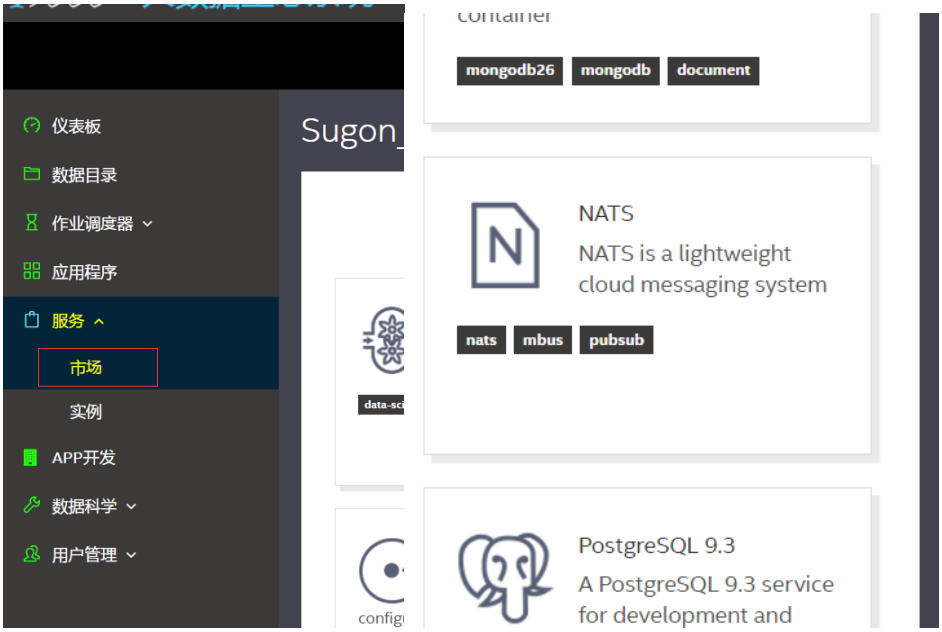
e) 应用打包部署流程优化；

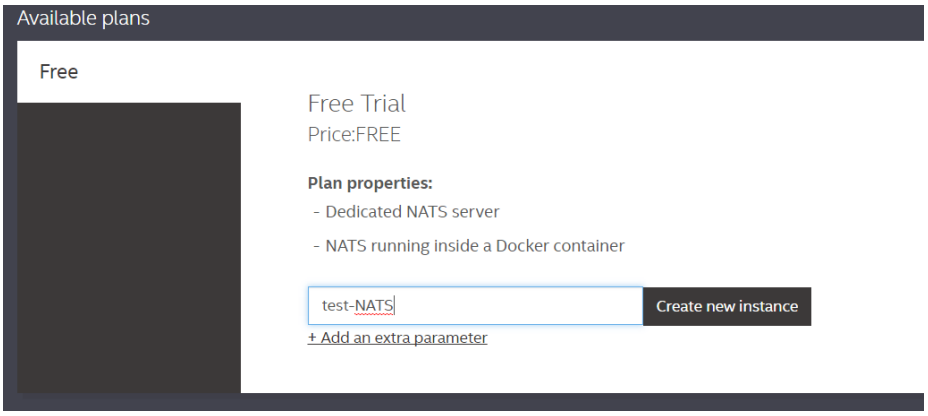
f) 同组件的不同实例分别部署到不同物理机的云主机上；

g) 其他组件功能优化。

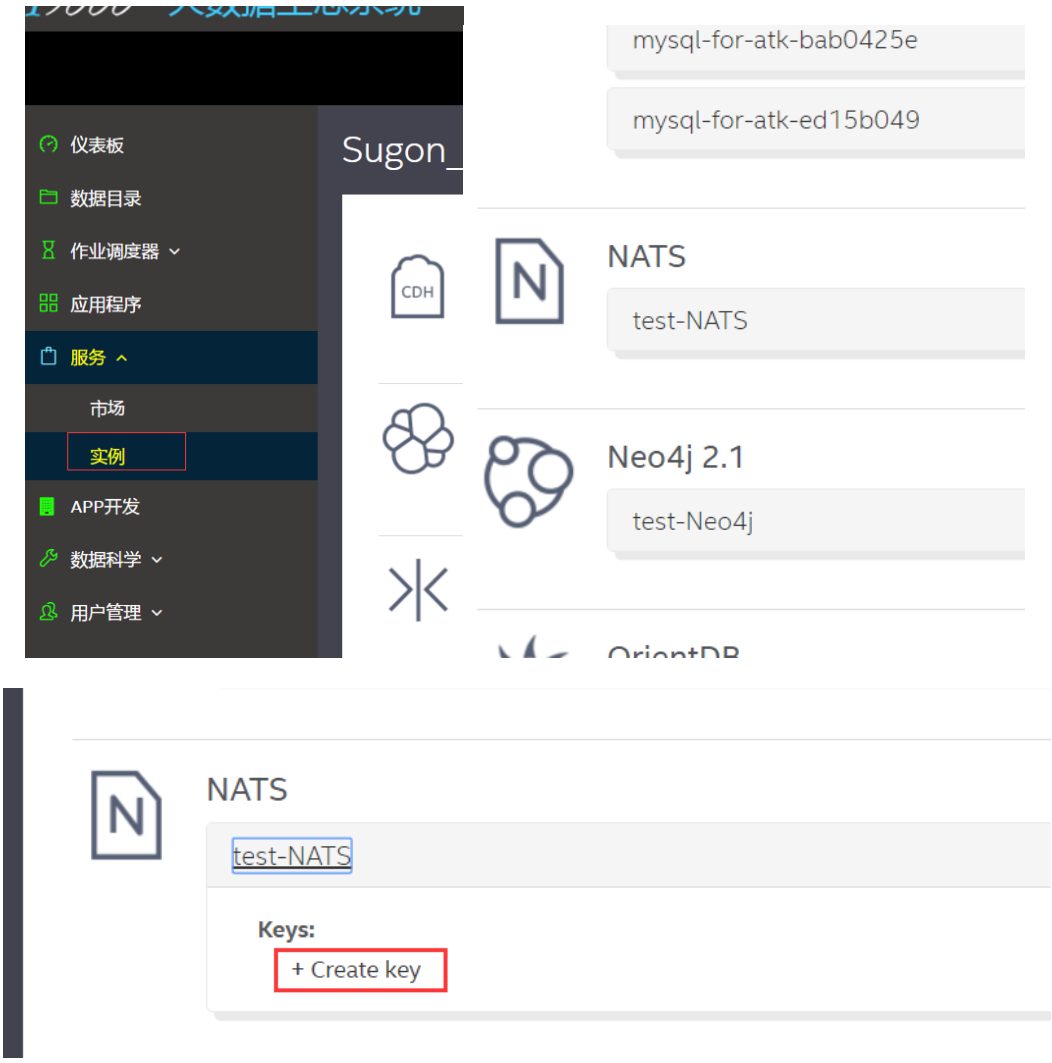
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 NATS，点击 NATS，创建一个新的实例。

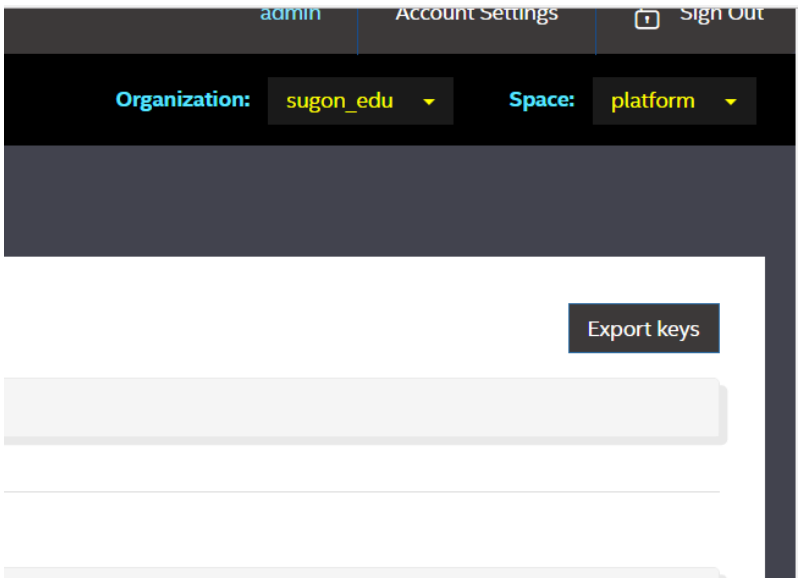




2. 在 服务>实例 里面找到 NATS，找到刚刚创建的实例，点击，创建新的 key。



3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 NATS 的连接信息。



NATS

test-NATS

Keys:

key + Add to exports

exported Keys:


```
{
  "nats": [
    {
      "label": "nats",
      "name": "test-NATS",
      "plan": "free",
      "tags": [
        "nats",
        "mbus",
        "pubsub"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "4222/tcp": "32800",
          "8333/tcp": "32801"
        },
        "port": "32800",
        "username": "zjmbqr1gptptzuuf",
        "password": "opjq5a8epwwxx9lk",
        "uri": "nats://zjmbqr1gptptzuuf:opjq5a8epwwxx9lk@10.0.4.4:32800"
      }
    }
  ]
}
```

4. 可用 NATS 官方提供的客户端或者其他应用中进行连接。

> 官方提供的客户端
>
> - Go client：https://github.com/nats-io/nats
> - Node.js client：https://github.com/nats-io/node-nats
> - Ruby client：https://github.com/nats-io/ruby-nats
> - Java client：https://github.com/nats-io/jnats
> - C client：https://github.com/nats-io/cnats
> - C# client：https://github.com/nats-io/csnats
> - Nginx C client：https://github.com/nats-io/nginx-nats

Neo4j 2.1

概述：

Neo4j 是一个高性能的,NOSQL 图形数据库，它将结构化数据存储在网络上而不是表中。它是一个嵌入式的、基于磁盘的、具备完全的事务特性的 Java 持久化引擎，但是它将结构化数据存储在网络(从数学角度叫做图)上而不是表中。Neo4j 也可以被看作是一个高性能的图引擎，该引擎具有成熟数据库的所有特性。程序员工作在一个面向对象的、灵活的网络结构下而不是严格、静态的表中——但是他们可以享受到具备完全的事务特性、企业级的数据库的所有好处。

Neo 是一个网络——面向网络的数据库——也就是说，它是一个嵌入式的、基于磁盘的、具备完全的事务特性的 Java 持久化引擎，但是它将结构化数据存储在网络上而不是表中。网络（从数学角度叫做图）是一个灵活的数据结构，可以应用更加敏捷和快速的开发模式。

特点：

1.对象关系的不匹配使得把面向对象的“圆的对象”挤到面向关系的“方的表”中是那么的困难和费劲，而这一切是可以避免的。

2.关系模型静态、刚性、不灵活的本质使得改变 schemas 以满足不断变化的业务需求是非常困难的。由于同样的原因，当开发小组想应用敏捷软件开发时，数据库经常拖后腿。

3.关系模型很不适合表达半结构化的数据——而业界的分析家和研究者都认为半结构化数据是信息管理中的下一个重头戏。

4.网络是一种非常高效的数据存储结构。人脑是一个巨大的网络，万维网也同样构造成

网状，这些都不是巧合。关系模型可以表达面向网络的数据，但是在遍历网络并抽取信息的能力上关系模型是非常弱的。

虽然 Neo 是一个比较新的开源项目，但它已经在具有 1 亿多个节点、关系和属性的产品中得到了应用，并且能满足企业的健壮性和性能的需求：

完全支持 JTA 和 JTS、2PC 分布式 ACID 事务、可配置的隔离级别和大规模、可测试的事务恢复。这些不仅仅是口头上的承诺：Neo 已经应用在高请求的 24/7 环境下超过 3 年了。它是成熟、健壮的，完全达到了部署的门槛。

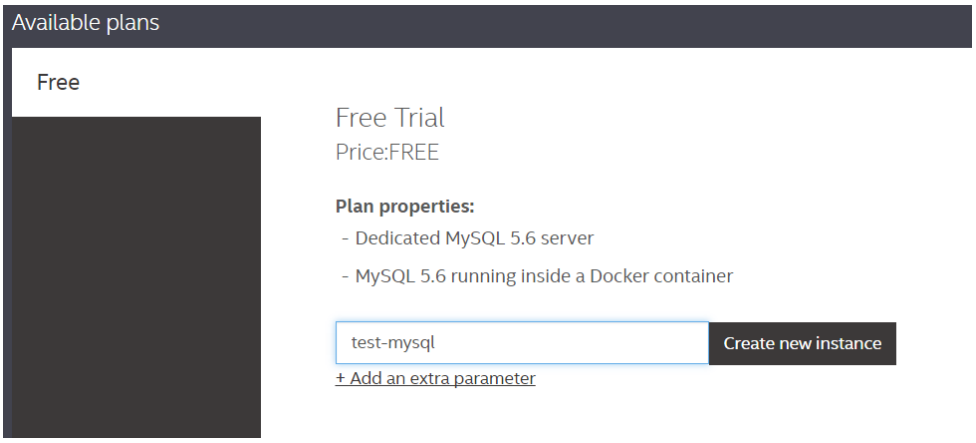
Neo4j 是一个嵌入式，基于磁盘的，支持完整事务的 Java 持久化引擎，它在图(网络)中而不是表中存储数据。Neo4j 提供了大规模可扩展性，在一台机器上可以处理数十亿节点/关系/属性的图，可以扩展到多台机器并行运行。相对于关系数据库来说，图数据库善于处理大量复杂、互连接、低结构化的数据，这些数据变化迅速，需要频繁的查询——在关系数据库中，这些查询会导致大量的表连接，因此会产生性能上的问题。Neo4j 重点解决了拥有大量连接的传统 RDBMS 在查询时出现的性能衰退问题。通过围绕图进行数据建模，Neo4j 会以相同的速度遍历节点与边，其遍历速度与构成图的数据量没有任何关系。此外，Neo4j 还提供了非常快的图算法、推荐系统和 OLAP 风格的分析，而这一切在目前的 RDBMS 系统中都是无法实现的。

由于使用了“面向网络的数据库”，人们对 Neo 充满了好奇。在该模型中，以“节点空间”来表达领域数据——相对于传统的模型表、行和列来说，节点空间是很多节点、关系和属性（键值对）构成的网络。关系是第一级对象，可以由属性来注解，而属性则表明了节点交互的上下文。网络模型完美的匹配了本质上就是继承关系的问题域，例如语义 Web

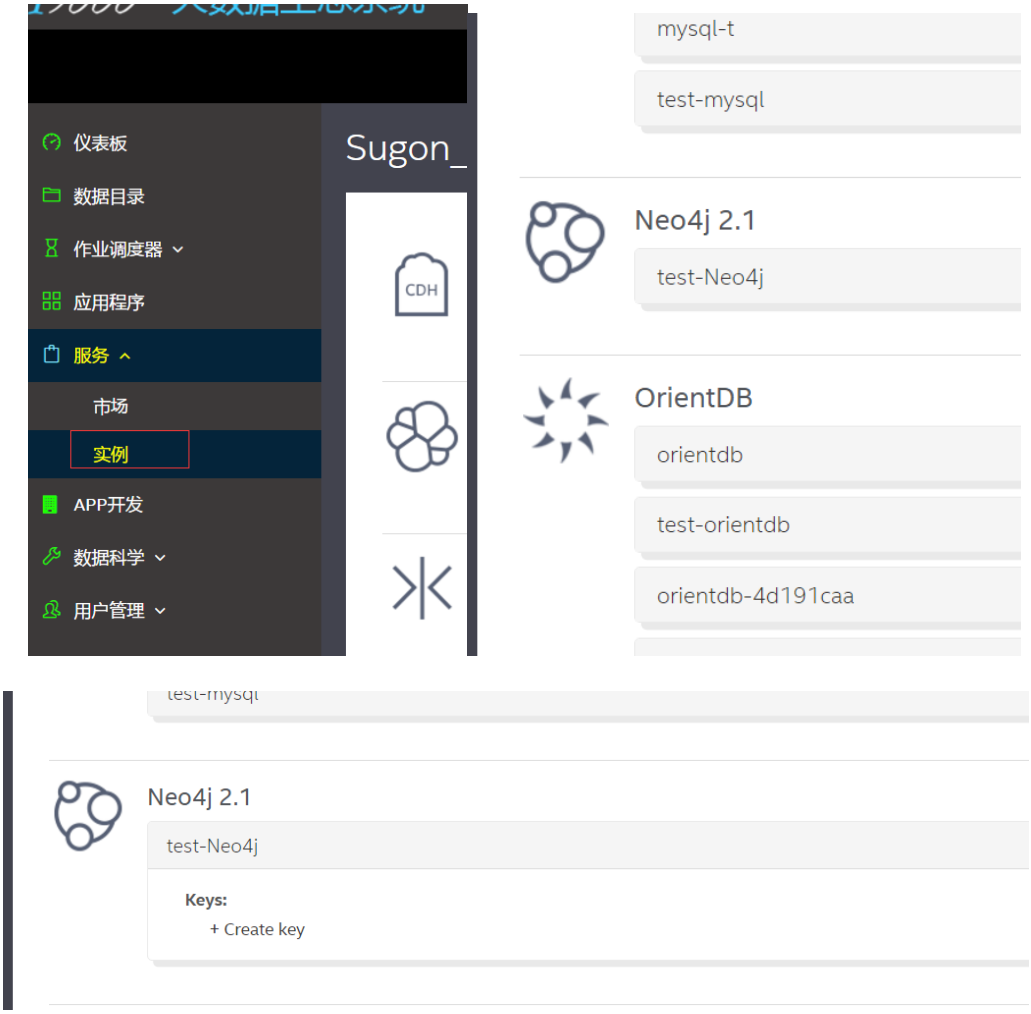
应用。

使用方法：

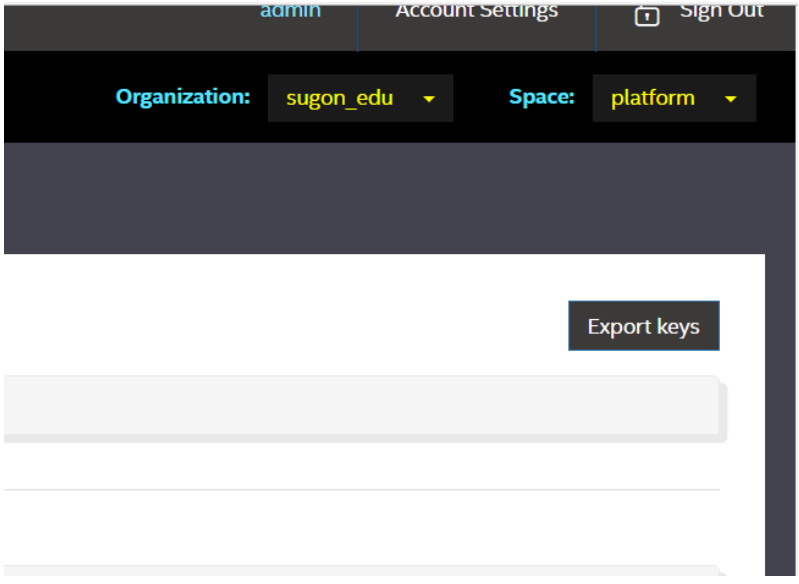
1. 在 i9000 平台上选择 服务>市场，找到 Neo4j 2.1，点击 Neo4j 2.1，创建一个新的实例。

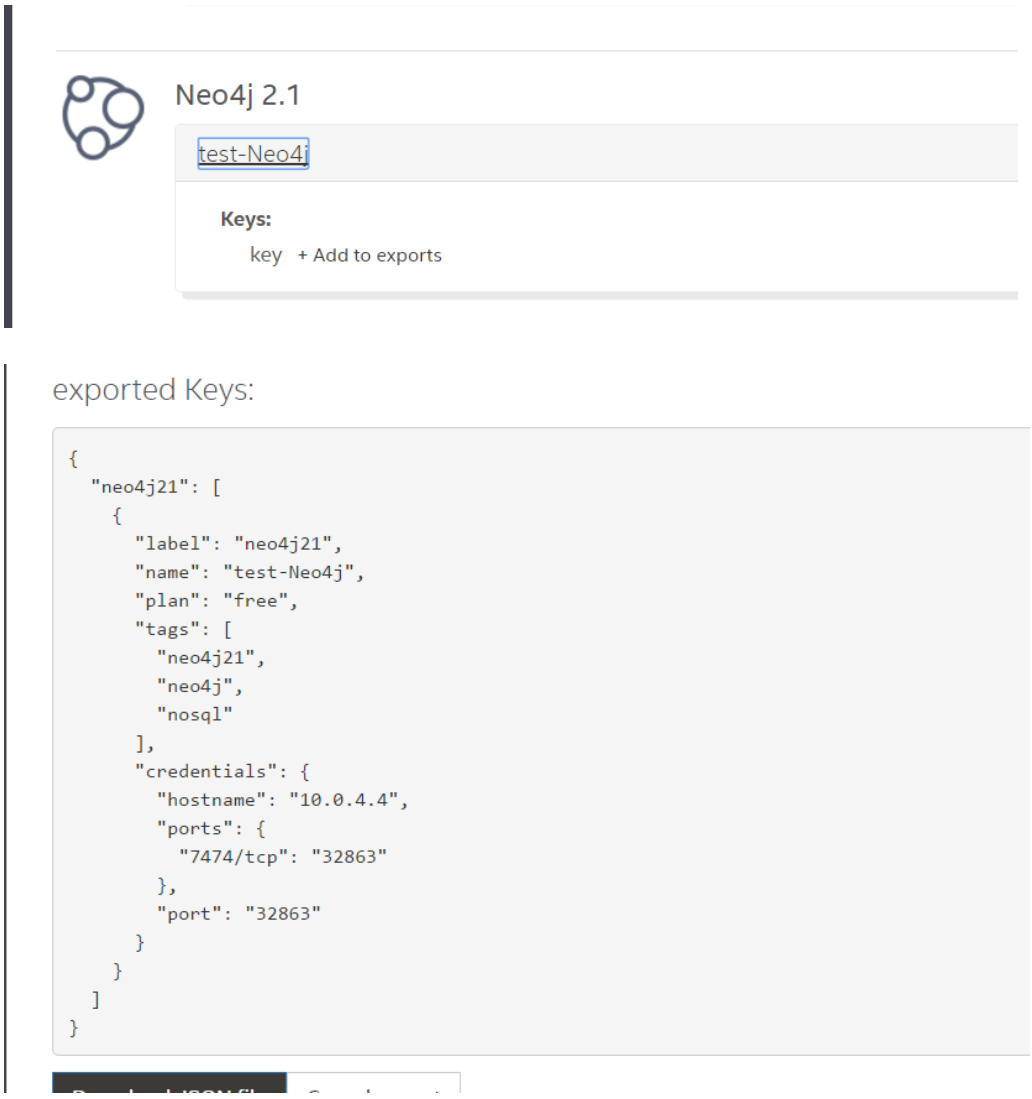


2. 在 服务>实例 里面找到 Neo4j 2.1，找到刚刚创建的实例，点击，创建新的 key。

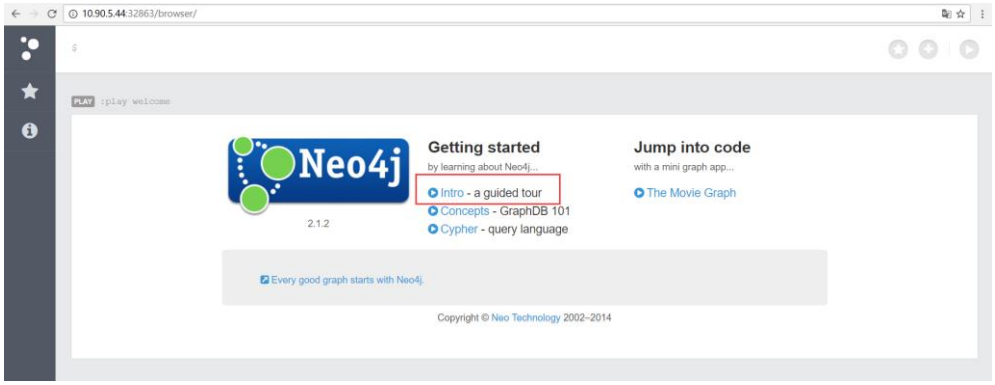


3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Neo4j 2.1 的连接信息。





4. 在浏览器里，可以通过 hostname，port 连接到刚刚创建的 Neo4j 2.1 数据库。



点击 guided tour 可以查看使用指南。

OrientDB

概述：

OrientDB 是一个开源 NoSQL 数据库管理系统。 NoSQL 数据库提供了存储和检索 NO-关系或引用诸如文档数据或图形数据比表格数据等数据的非关系型数据的机制。NoSQL 数据库越来越多地用于大数据和实时 Web 应用程序。 NoSQL 系统有时也被称为“不只是 SQL”，以强调它们可能支持类似 SQL 的查询语言。

OrientDB 也属于 NoSQL 系列。 OrientDB 是第二代分布式数据库数据库，具有一个产品中的文档灵活性，具有 Apache 2 许可证的开放源代码。 在 OrientDB 之前市场上有几个 NoSQL 数据库，其中一个是 MongoDB。

MongoDB 和 OrientDB 包含许多常见功能，但引擎是根本不同的。 MongoDB 是纯文档数据库，OrientDB 是一个具有图表引擎的混合文档。

特征	MongoDB	OrientDB
关系	使用RDBMS JOINS创建实体之间的关系。 它具有高运行时成本，并且当数据库规模增加时不扩展。	嵌入和连接文档，如关系数据库。 它使用从图形数据库世界采取的直接，超快速链接。
提取计划	成本高的操作。	轻松返回带有互连文档的完整图形。
交易	不支持ACID事务，但它支持原子操作。	支持ACID事务以及原子操作。
查询语言	有自己的基于JSON的语言。	查询语言建立在SQL上。
索引	对所有索引使用B树算法。	支持三种不同的索引算法，使用户可以实现最佳性能。
存储引擎	使用内存映射技术。	使用存储引擎名称LOCAL和PLOCAL。

OrientDB 是第一个多模型开源 NoSQL DBMS，将图形的功能和文档的灵活性集成到

可扩展的高性能操作数据库中。

OrientDB 的分布式架构原理

OrientDB 是一个分布式的图结构加文档结构数据库，其 1.0 版本将在 12 月份发布，1.0 版本中最重大的改进就是支持 multi-master 的分布式结构，下面 PPT 就是 OrientDB 官方对其分布式结构原理的详细描述。

其主要特性包括下面几点：

支持 multi-master 的多点读写

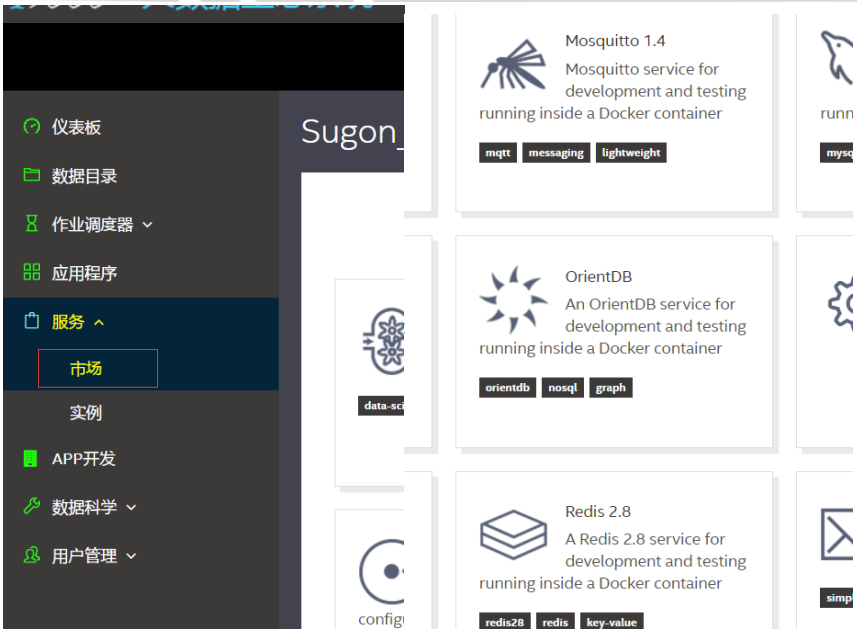
可以设置数据是同步复制还是异步复制

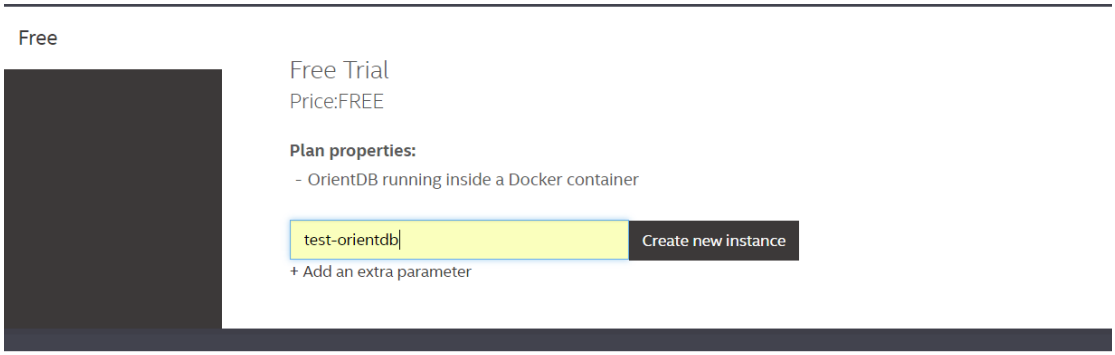
支持透明故障转移

对多点写造成的冲突支持自动解决和手动解决两种方式

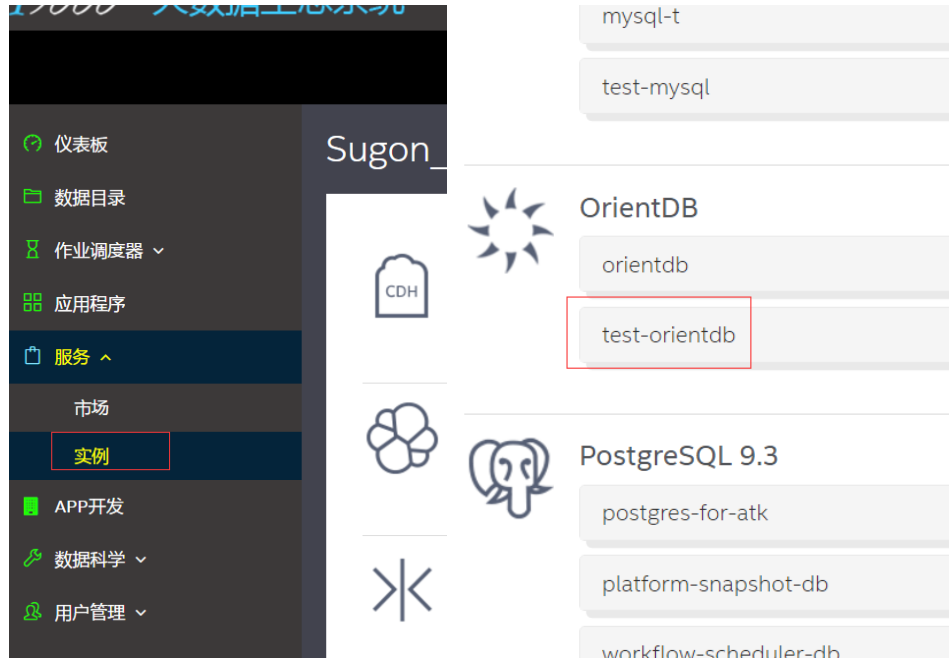
使用方法：

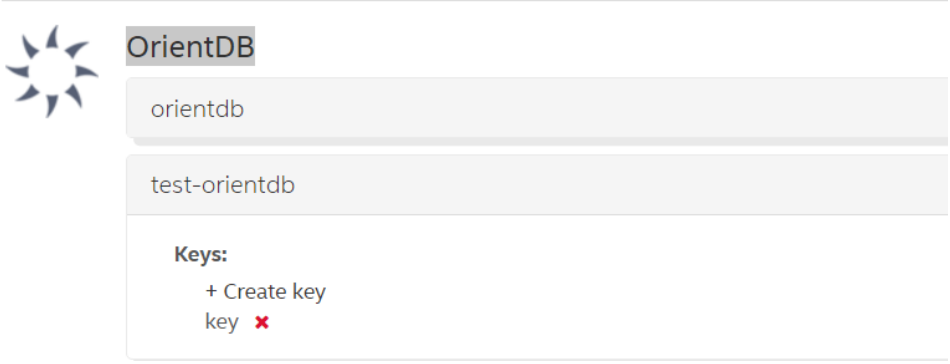
1. 在 i9000 平台上选择 服务>市场，找到 Orient DB，点击 Orient DB，创建一个新的实例。



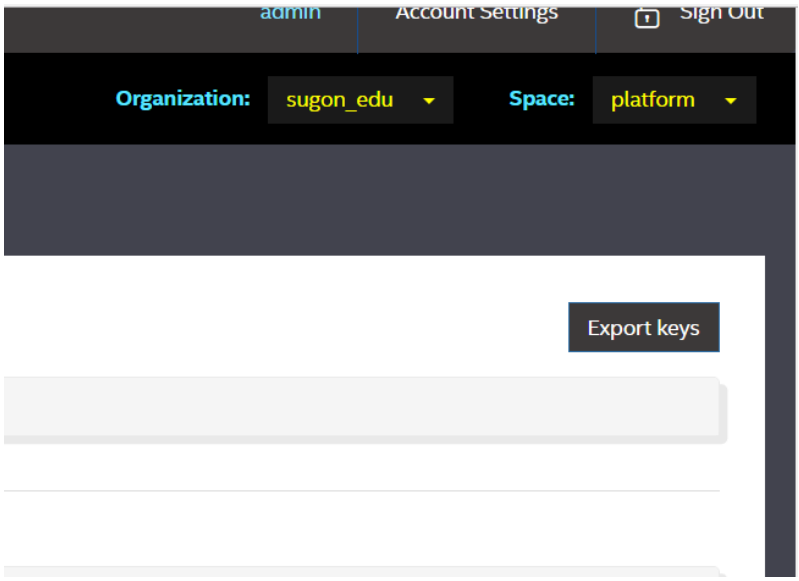


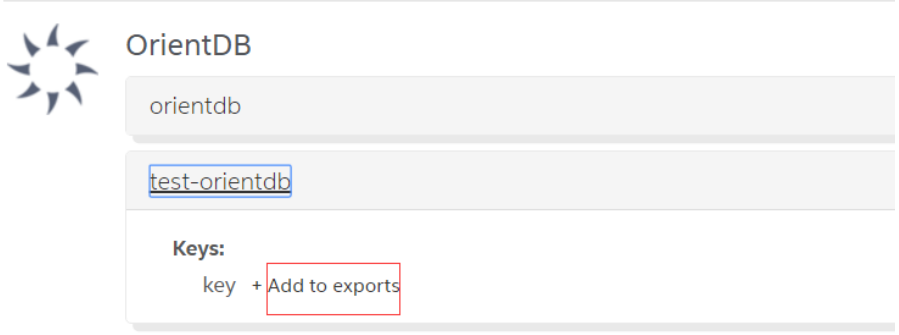
2. 在 服务>实例 里面找到 Orient DB，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Orient DB 的连接信息。

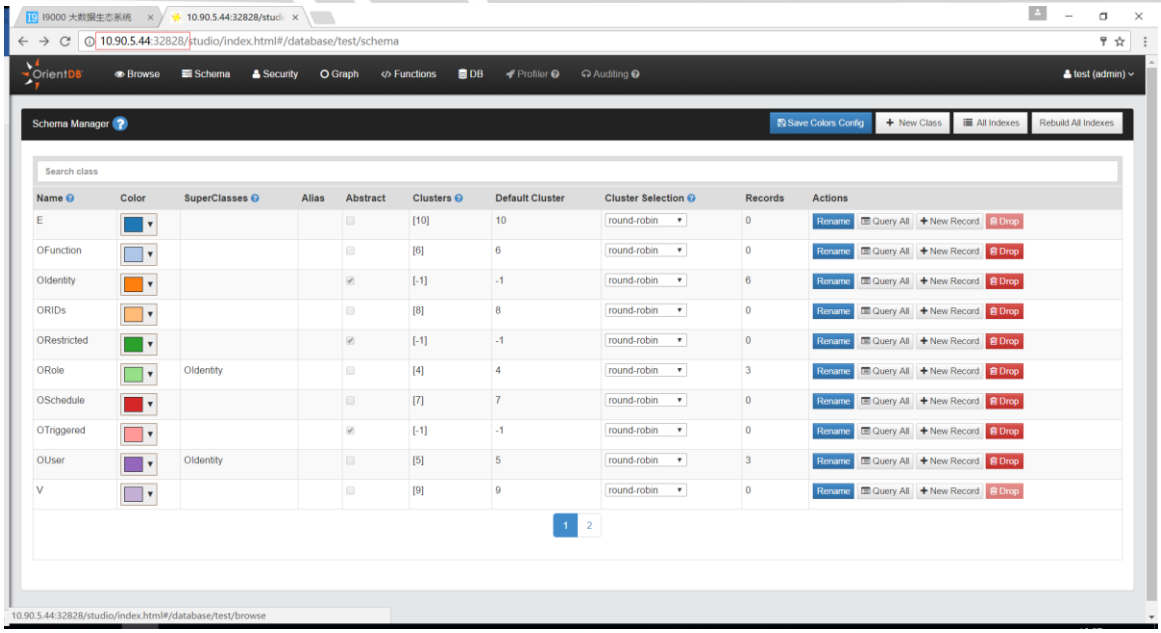




exported Keys:


```
{
  "orientdb": [
    {
      "label": "orientdb",
      "name": "test-orientdb",
      "plan": "free",
      "tags": [
        "orientdb",
        "nosql",
        "graph"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "2424/tcp": "32827",
          "2480/tcp": "32828"
        },
        "password": "4ihwqi59rtduzlmc"
      }
    }
  ]
}
```


4. 在浏览器里输入新建实例的 ip 地址和端口号(注意是 2480 端口映射的端口号)进行访问，可以看到界面化的 Orient DB。


10.90.5.44:32828/studio/index.html#/database/test/schema
OrientDB
Browse
Schema
Security
Graph
Functions
DB
Profiler
Auditing
test (admin)
Schema Manager
Save Colors Config
New Class
All Indexes
Rebuild All Indexes
Search class
Name
Color
SuperClasses
Alias
Abstract
Clusters
Default Cluster
Cluster Selection
Records
Actions
E [10] 10 round-robin 0
OFunction [6] 6 round-robin 0
OIdentity [-1] -1 round-robin 6
ORIDs [8] 8 round-robin 0
ORestricted [-1] -1 round-robin 0
ORole OIdentity [4] 4 round-robin 3
OSchedule [7] 7 round-robin 0
OTriggered [-1] -1 round-robin 0
OUser OIdentity [5] 5 round-robin 3
V [9] 9 round-robin 0
Rename
Query All
New Record
Drop
1
2
10.90.5.44:32828/studio/index.html#/database/test/browse

OrientDB Dashboard

OrientDB 企业版包括一个新的易于阅读和单页面的仪表板，并提供经过更新的报表。仪表板以图形方式显示加入集群的每个节点的当前状态和历史趋势。 报告绩效指标，以便能够即时做出知情决策，您可以一目了然。

在这里，您可以看到仪表板报告由两个节点组成的集群的状态。

对于每个节点，您可以监视分为两个主要部分的几个信息：

系统报告

CPU ， RAM ， DISK CACHE 和 DISK 使用

Status 的节点

Operations per second

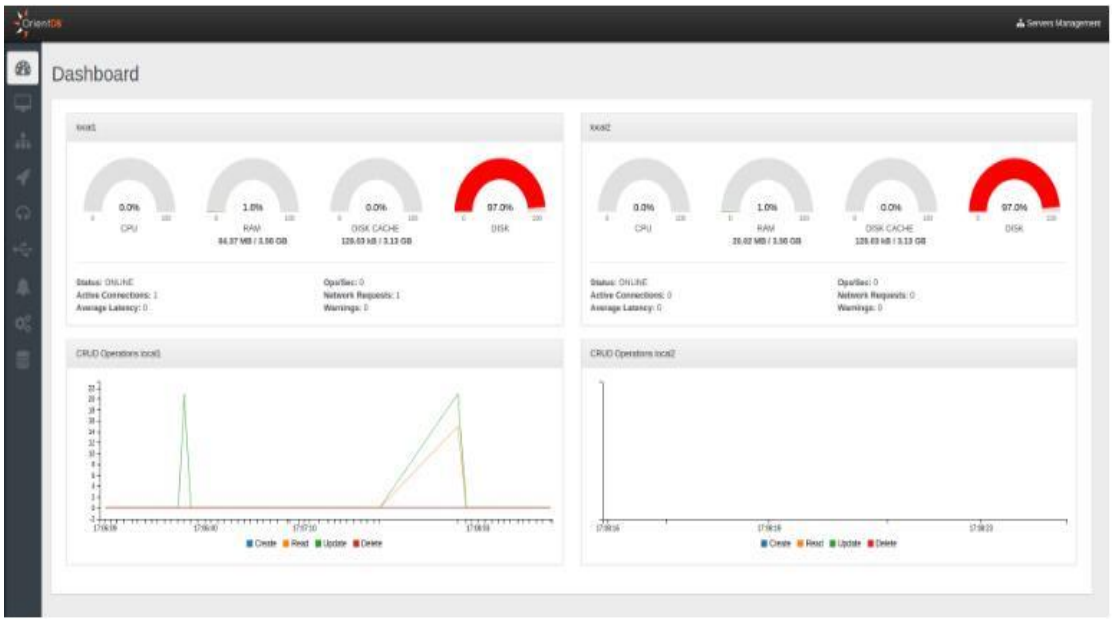
Active Connections

Network Request

Average Latency

Warnings

CRUD 操作：包括 Live Chart CRUD 操作的实时性。



这是节（仅适用于企业版）与 OrientDB 服务器一起使用作为 DBA / DevOps。 这从 OrientDB 2.1 工作室未来的控制面板已经丰富了新几项新功能企业版 。

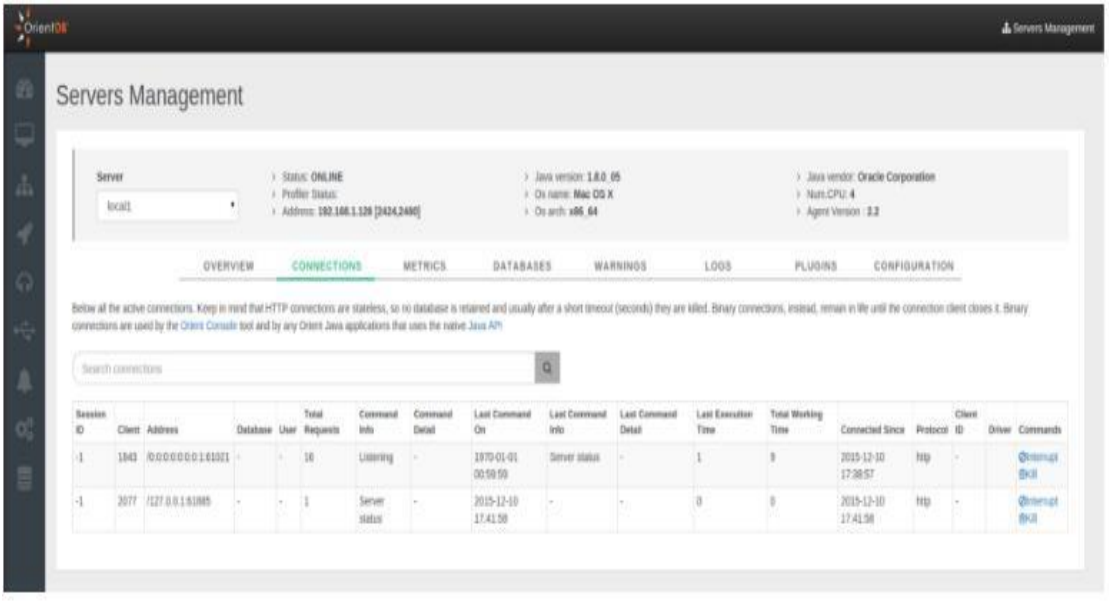
在页面顶部，您可以选择您的服务器，可视化其系统信息，然后通过可用的选项卡导航与其相关的所有统计信息和事实。

连接：

它显示到服务器的所有活动连接。 对于每个连接，报告以下信息：

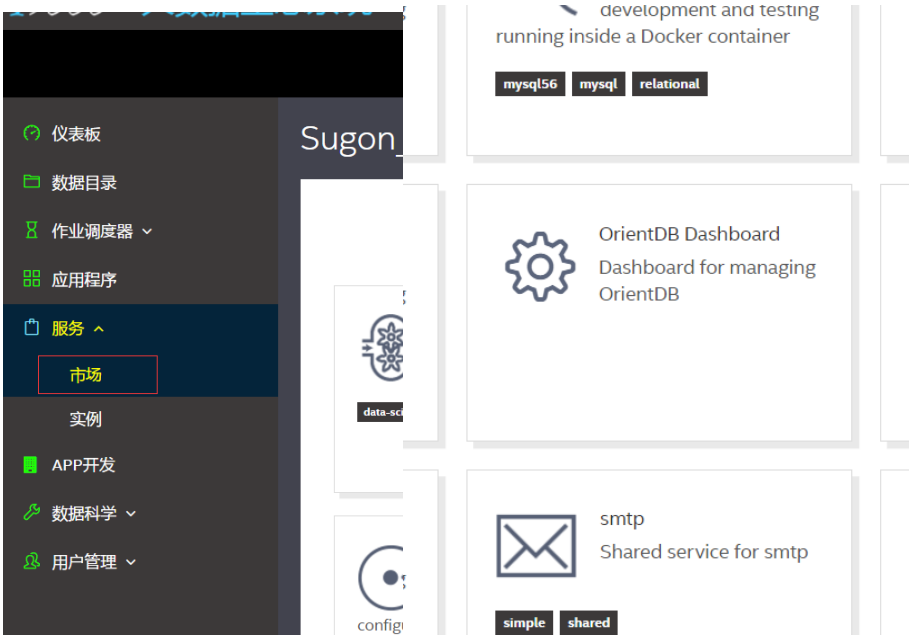
- Session ID ，作为唯一的会话号
- Client ，作为唯一的客户端数
- Address ，是连接源
- Database ，使用数据库名
- User ，数据库用户
- Total Requests ，如要求由连接执行的总数
- Command Info ，如运行命令

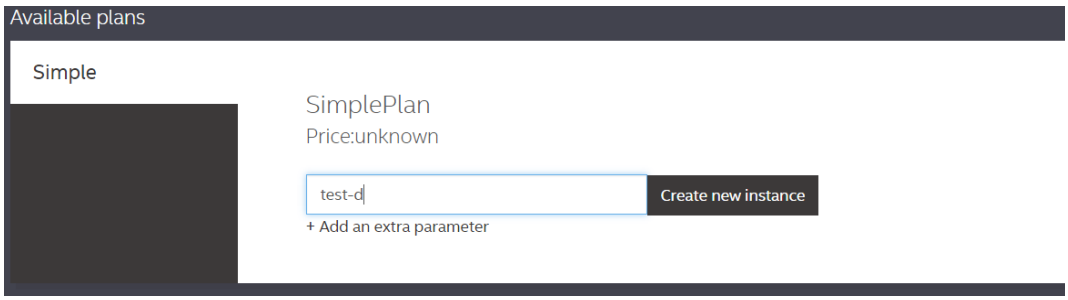
- Command Detail ，大约运行命令的详细信息
- Last Command On ，是一个请求已执行的最后一次
- Last Command Info ，是关于最后的操作执行，情报
- Last Command Detail ，是关于上次操作的执行细节，情报
- Last Execution Time ，是执行时间为 O 最后一个请求
- Total Working Time ，是迄今为止由当前连接所用的总执行时间
- Connected Since ，是在连接已被创建时的日期
- Protocol ，是其中协议 HTTP 和二进制
- Client ID ，代表客户端连接的文本
- Driver ，驱动程序名称
- Commands ，命令按钮， Interrupt 或 Kill 每个会话。



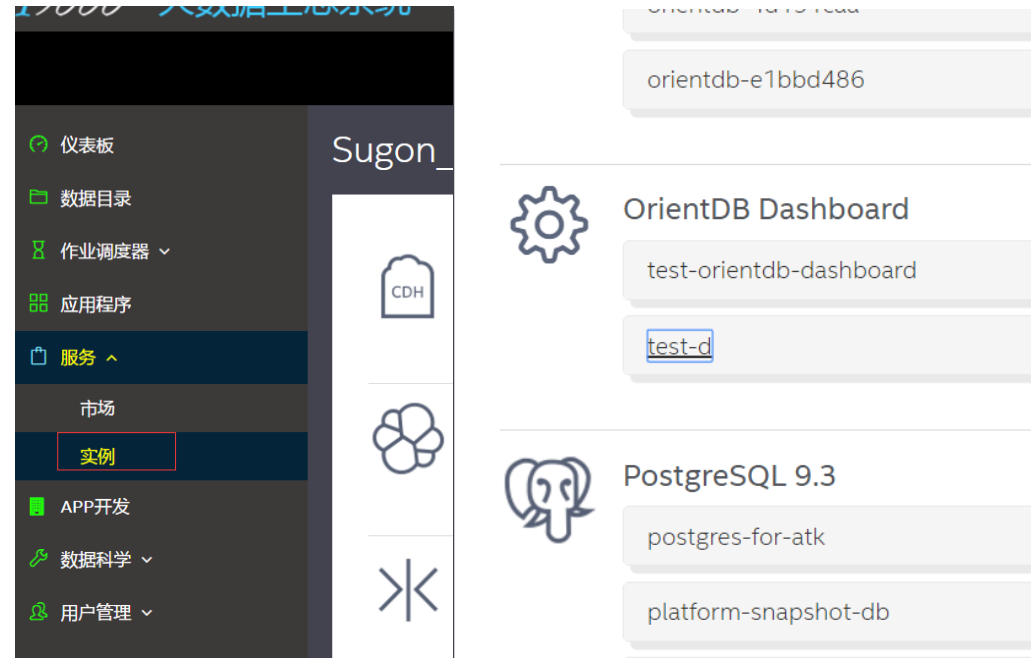
使用方法：

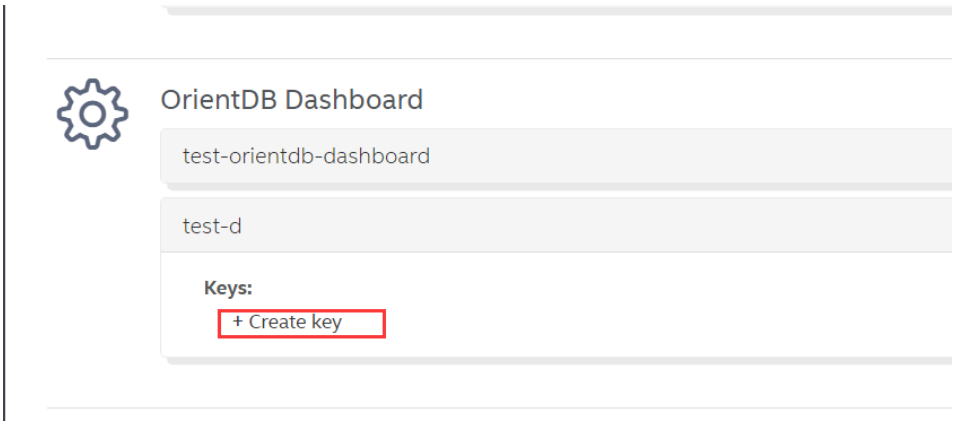
1. 在 i9000 平台上选择 服务>市场，找到 OrientDB Dashboard，点击 Orient DB Dashboard，创建一个新的实例。



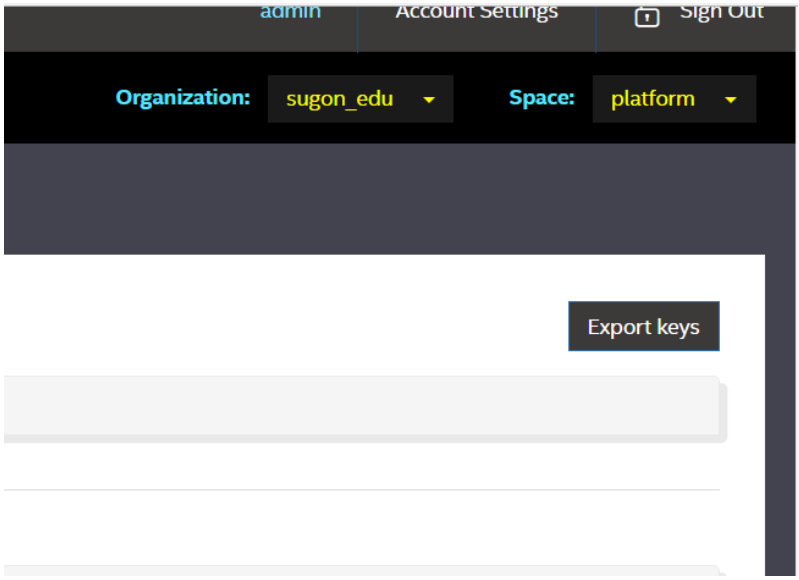


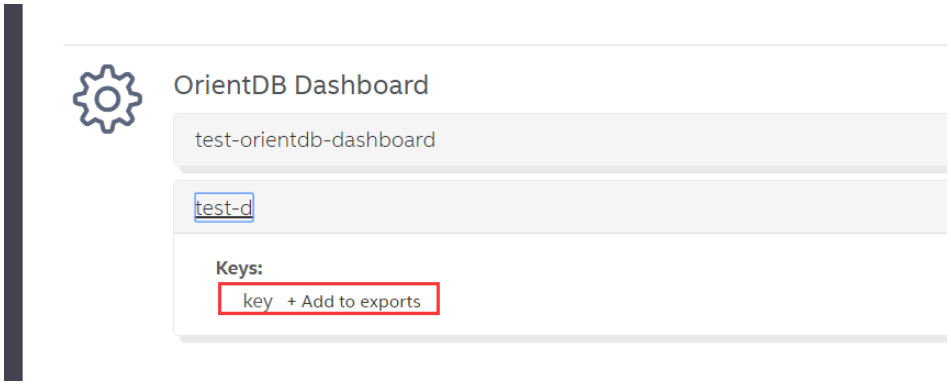
2. 在 服务>实例 里面找到 OrientDB Dashboard，找到刚刚创建的实例，点击，创建新的 key。



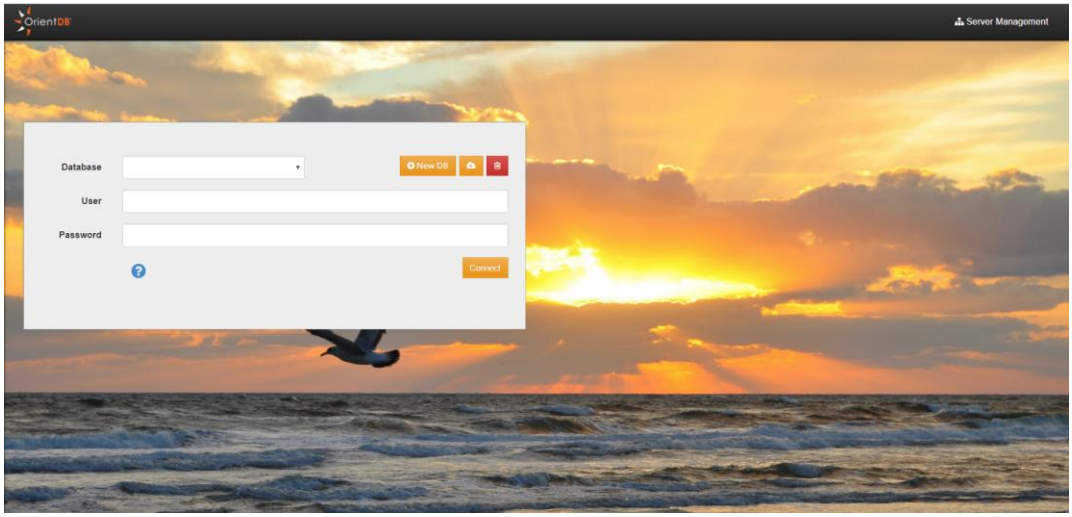


3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 OrientDB Dashboard 的连接信息。





4. 在浏览器里输入新建实例的 url，即可访问新建的 OrientDB Dashboard。


OrientDB
Server Management
Database
New DB
User
Password
Connect

PostgreSQL 9.3

概述：

PostgreSQL 是以加州大学伯克利分校计算机系开发的 POSTGRES，现在已经更名为 PostgreSQL，版本 4.2 为基础的对象关系型数据库管理系统（ORDBMS）。PostgreSQL 支持大部分 SQL 标准并且提供了许多其他现代特性：复杂查询、外键、触发器、视图、事务完整性、MVCC。同样，PostgreSQL 可以用许多方法扩展，比如， 通过增加新的数据类型、函数、操作符、聚集函数、索引。免费使用、修改、和分发 PostgreSQL，不管是私用、商用、还是学术研究使用。

优缺点：

PostgreSQL 是一个自由的对象-关系数据库服务器(数据库管理系统)，它在灵活的 BSD-风格许可证下发行。它提供了相对其他开放源代码数据库系统(比如 MySQL 和 Firebird)，和专有系统(比如 Oracle、Sybase、IBM 的 DB2 和 Microsoft SQL Server)之外的另一种选择。

PostgreSQL 不寻常的名字导致一些读者停下来尝试拼读它，特别是那些把 SQL 拼读为"sequel"的人。PostgreSQL 开发者把它拼读为 "post-gress-Q-L"。它也经常被简略念为 "postgres"。

优点

事实上， PostgreSQL 的特性覆盖了 SQL-2/SQL-92 和 SQL-3/SQL-99，首先，它包括了可以说是目前世界上最丰富的数据类型的支持 ,其中有些数据类型可以说连商业数据库都不具备， 比如 IP 类型和几何类型等；其次，PostgreSQL 是全功能的自由软件数据

库，很长时间以来，PostgreSQL 是唯一支持事务、子查询、多版本并行控制系统（MVCC）、数据完整性检查等特性的唯一的一种自由软件的数据库管理系统。 Inprise 的 InterBase 以及 SAP 等厂商将其原先专有软件开放为自由软件之后才打破了这个唯一。最后，PostgreSQL 拥有一支非常活跃的开发队伍，而且在许多黑客的努力下，PostgreSQL 的质量日益提高。

从技术角度来讲，PostgreSQL 采用的是比较经典的 C/S（client/server）结构，也就是一个客户端对应一个服务器端守护进程的模式，这个守护进程分析客户端来的查询请求，生成规划树，进行数据检索并最终把结果格式化输出后返回给客户端。为了便于客户端的程序的编写，由数据库服务器提供了统一的客户端 C 接口。而不同的客户端接口都是源自这个 C 接口，比如 ODBC，JDBC，Python，Perl，Tcl，C/C++，ESQL 等， 同时也要指出的是，PostgreSQL 对接口的支持也是非常丰富的，几乎支持所有类型的数据库客户端接口。这一点也可以说是 PostgreSQL 一大优点。

缺点

从 Postgres 开始，PostgreSQL 就经受了多次变化。

首先，早期的 PostgreSQL 继承了几乎所有 Ingres, Postgres, Postgres95 的问题：过于学院味，因为首先它的目的是数据库研究，因此不论在稳定性， 性能还是使用方方面面，长期以来一直没有得到重视，直到 PostgreSQL 项目开始以后，情况才越来越好，PostgreSQL 已经完全可以胜任任何中上规模范围内的应用范围的业务。目前有报道的生产数据库的大小已经有 TB 级的数据量，已经逼近 32 位计算的极限。不过学院味也给 PostgreSQL 带来一个意想不到的好处：大概因为各大学的软硬件环境差异太大的缘故，它

是目前支持平台最多的数据库管理系统的一种 ,所支持的平台多达十几种 ,包括不同的系统，不同的硬件体系。至今，它仍然保持着支持平台最多的数据库管理系统的称号。

其次 , PostgreSQL 的确还欠缺一些比较高端的数据库管理系统需要的特性 , 比如数据库集群，更优良的管理工具和更加自动化的系统优化功能 等提高数据库性能的机制等。

PostgreSQL 数据库架构：

PostgreSQL 强壮的一个原因源于它的架构。和商业数据库一样，PostgreSQL 可以用于 C/S(客户/服务器)环境。这对于用户和开发人员有很多好处。

PostgreSQL 安装核心是数据库服务端进程。它允许在一个独立服务器上。需要访问存储在数据库中的数据的应用程序必须通过数据库进程。这些客户端程序无法直接访问数据，即使它们和服务程序在同一台机器上。

注 : PostgreSQL 还不具有一些企业级商业数据库的负载均衡和提供扩展的可伸缩性和可恢复性的 HA(High-Availability , 高可用性)功能。在 http://gborg.postgresql.org 有一些 PostgreSQL 认可的项目针对增加这些功能在进行中，同时也有一些商业解决方案存在。

这种分开为客户端和服务器端的方式可以让应用程序分布式允许。你可以使用网络来分隔你的客户端和服务端，使开发的客户端程序适合用户的使用环境。例如，你可以在 UNIX 上实现数据库并建立运行在在 Microsoft Windows 上的客户端程序。

服务器专注于数据处理 ,而不是尝试控制很多客户端访问服务器上共享目录中存储的数据，这让 PostgreSQL 高效的管理数据的完整性，即使在存在大量的用户的情况下。

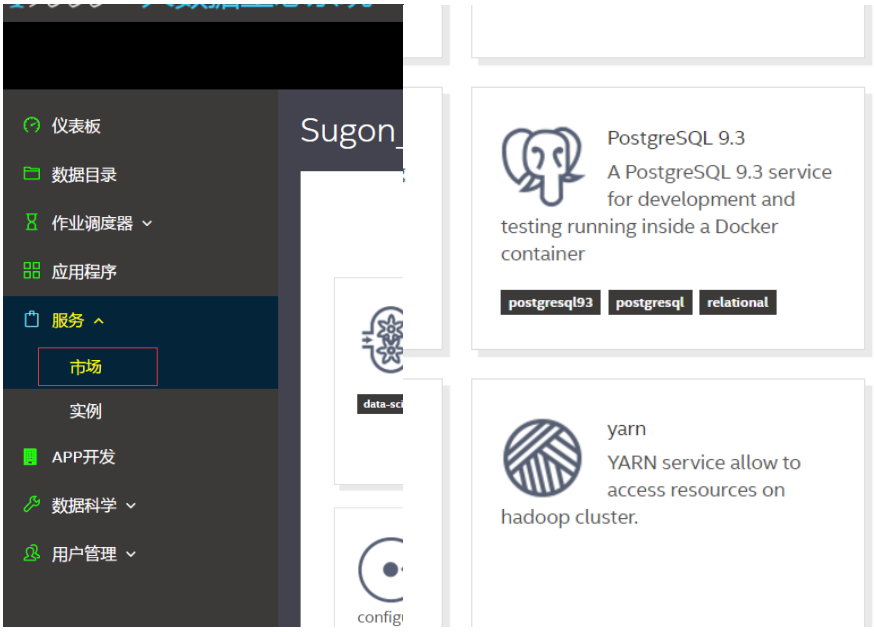
客户端程序使用专有协议连接到 PostgreSQL。然而，通过在客户端安装软件而为应用程序提供一个标准的接口是可能的，例如开放式数据库连接(ODBC，Open Database

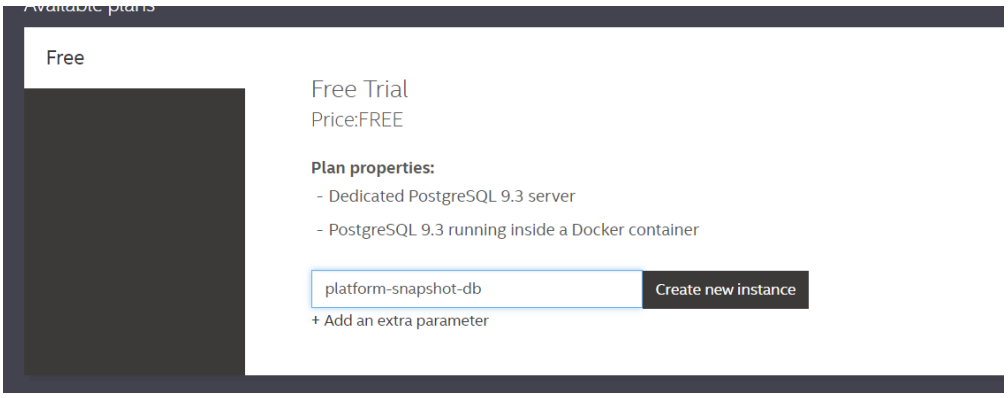
Connectivity)标准或者 Java 程序的 Java 数据库连接(JDBC)标准。ODBC 驱动的存在运行很现存的应用程序使用 PostgreSQL 作为一个数据库 ,包括的微软的 Office 产品例如 Excel 和 Access。你将在第三、第五和第十三到十八章了解不同的 PostgreSQL 连接方法。

PostgreSQL 的 C/S 架构允许任务分工。非常适合于存储和访问大量数据的服务器主机可以用作安全的数据储存库。可以为客户端开发复杂的图形界面程序。另外，基于网页的前端可以通过建立网页模式的结果集到浏览器访问数据，而不需要另外的客户端软件。我们将在第五章和十五章回头讨论这些想法。

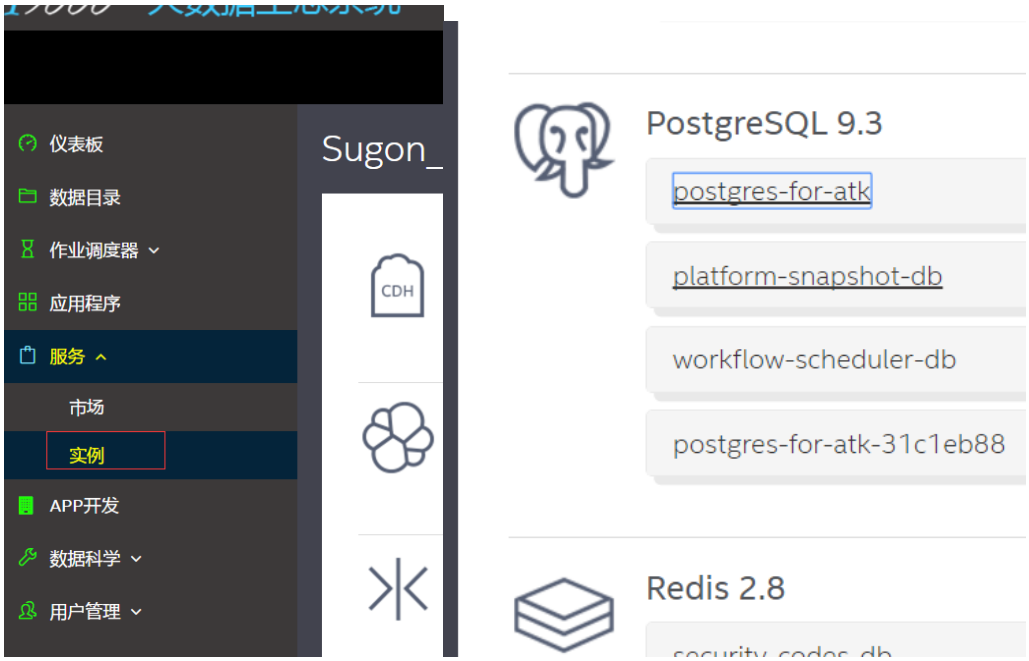
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 PostgreSQL 9.3，点击 PostgreSQL 9.3，创建一个新的实例。

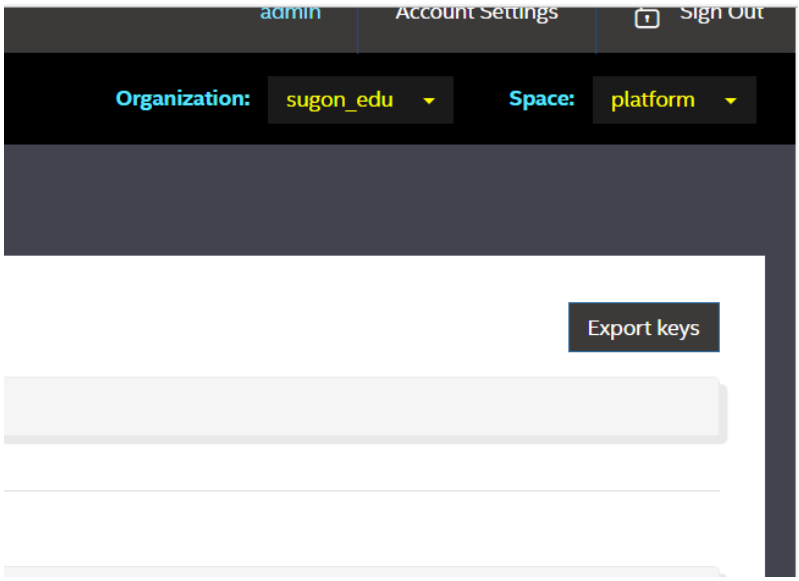


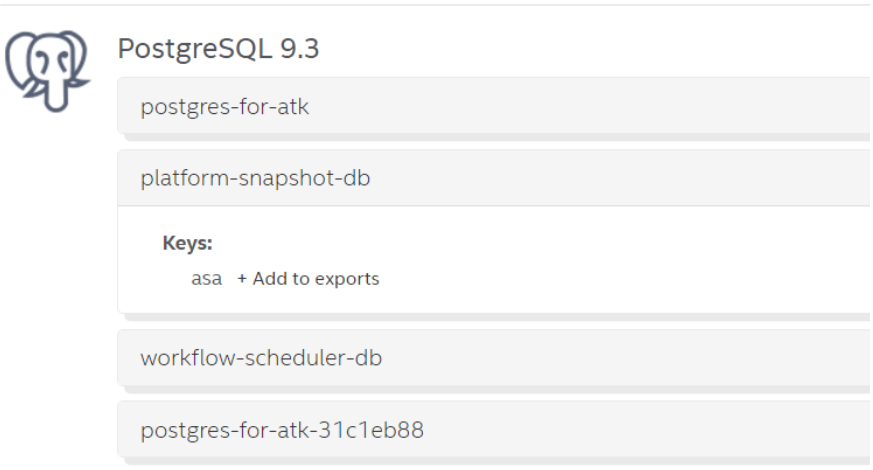


2. 在 服务>实例 里面找到 PostgreSQL 9.3，找到刚刚创建的实例，点击，创建新的 key。



3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 PostgreSQL 9.3 的连接信息。





exported Keys:


```
{
  "postgresql93": [
    {
      "label": "postgresql93",
      "name": "platform-snapshot-db",
      "plan": "free",
      "tags": [
        "postgresql93",
        "postgresql",
        "relational"
      ],
      "credentials": {
        "hostname": "10.0.4.4",
        "ports": {
          "5432/tcp": "32768"
        },
        "port": "32768",
        "username": "sajvof7ehwkzgtxf",
        "password": "flvpzihngyzdlfb7",
        "dbname": "tcjtq9vydc1xxpws",
        "uri": "postgres://sajvof7ehwkzgtxf:flvpzihngyzdlfb7@10.0.4.4:32768/tcjtq9vydc1xxpws"
      }
    }
  ]
}
```

4. 使用可视化客户端工具，输入 hostname，port，username，password 等信息即可连接到数据库。

Yarn

Yarn 简介：

Apache Hadoop YARN （Yet Another Resource Negotiator，另一种资源协调者）是一种新的 Hadoop 资源管理器，它是一个通用资源管理系统，可为上层应用提供统一的资源管理和调度，它的引入为集群在利用率、资源统一管理和数据共享等方面带来了巨大好处。

YARN 的基本思想是将 JobTracker 的两个主要功能(资源管理和作业调度/监控)分离，主要方法是创建一个全局的 ResourceManager（ RM ）和若干个针对应用程序的 ApplicationMaster（ AM ）。这里的应用程序是指传统的 MapReduce 作业或作业的 DAG（有向无环图）。

YARN 分层结构的本质是 ResourceManager。这个实体控制整个集群并管理应用程序向基础计算资源的分配。ResourceManager 将各个资源部分（计算、内存、带宽等）精心安排给基础 NodeManager（YARN 的每节点代理）。ResourceManager 还与 ApplicationMaster 一起分配资源，与 NodeManager 一起启动和监视它们的基础应用程序。在此上下文中，ApplicationMaster 承担了以前的 TaskTracker 的一些角色，ResourceManager 承担了 JobTracker 的角色。

ApplicationMaster 管理一个在 YARN 内运行的应用程序的每个实例。ApplicationMaster 负责协调来自 ResourceManager 的资源，并通过 NodeManager 监视容器的执行和资源使用（CPU、内存等的资源分配）。请注意，尽管目前的资源更加传统（CPU 核心、内存），但未来会带来基于手头任务的新资源类型（比如图形处理单元或专用处理设备）。从 YARN 角度讲，ApplicationMaster 是用户代码，因此存在潜在的安

全问题。YARN 假设 ApplicationMaster 存在错误或者甚至是恶意的，因此将它们当作无特权的代码对待。

NodeManager 管理一个 YARN 集群中的每个节点。NodeManager 提供针对集群中每个节点的服务，从监督对一个容器的终生管理到监视资源和跟踪节点健康。MRv1 通过插槽管理 Map 和 Reduce 任务的执行，而 NodeManager 管理抽象容器，这些容器代表着可供一个特定应用程序使用的针对每个节点的资源。YARN 继续使用 HDFS 层。它的主要 NameNode 用于元数据服务，而 DataNode 用于分散在一个集群中的复制存储服务。

要使用一个 YARN 集群，首先需要来自包含一个应用程序的客户的请求。ResourceManager 协商一个容器的必要资源，启动一个 ApplicationMaster 来表示已提交的应用程序。通过使用一个资源请求协议，ApplicationMaster 协商每个节点上供应用程序使用的资源容器。执行应用程序时，ApplicationMaster 监视容器直到完成。当应用程序完成时，ApplicationMaster 从 ResourceManager 注销其容器，执行周期就完成了。

MRv1 的缺陷

MapReduce 的第一个版本既有优点也有缺点。MRv1 是目前使用的标准的大数据处理系统。但是，这种架构存在不足，主要表现在大型集群上。当集群包含的节点超过 4,000 个时（其中每个节点可能是多核的），就会表现出一定的不可预测性。其中一个最大的问题是级联故障，由于要尝试复制数据和重载活动的节点，所以一个故障会通过网络泛洪形式导致整个集群严重恶化。

但 MRv1 的最大问题是多租户。随着集群规模的增加，一种可取的方式是为这些集群采用各种不同的模型。MRv1 的节点专用于 Hadoop，所以可以改变它们的用途以用于其

他应用程序和工作负载。当大数据和 Hadoop 成为云部署中一个更重要的使用模型时，这种能力也会增强，因为它允许在服务器上对 Hadoop 进行物理化，而无需虚拟化且不会增加管理、计算和输入/输出开销。

Yarn 的优点

大大减小了 JobTracker（也就是现在的 ResourceManager）的资源消耗，并且让监测每一个 Job 子任务 (tasks) 状态的程序分布式化了，更安全、更优美。

在新的 Yarn 中，ApplicationMaster 是一个可变更的部分，用户可以对不同的编程模型写自己的 AppMst，让更多类型的编程模型能够跑在 Hadoop 集群中，可以参考 hadoop Yarn 官方配置模板中的 mapred-site.xml 配置。对于资源的表示以内存为单位（在目前版本的 Yarn 中，没有考虑 cpu 的占用），比之前以剩余 slot 数目更合理。

老的框架中，JobTracker 一个很大的负担就是监控 job 下的 tasks 的运行状况，现在，这个部分就扔给 ApplicationMaster 做了，而 ResourceManager 中有一个模块叫做 ApplicationsMasters(注意不是 ApplicationMaster)，它是监测 ApplicationMaster 的运行状况，如果出问题，会将其在其他机器上重启。

Container 是 Yarn 为了将来作资源隔离而提出的一个框架。这一点应该借鉴了 Mesos 的工作，目前是一个框架，仅仅提供 java 虚拟机内存的隔离,hadoop 团队的设计思路应该后续能支持更多的资源调度和控制，既然资源表示成内存量，那就没有了之前的 map slot/reduce slot 分开造成集群资源闲置的尴尬情况。

YARN 的核心思想

将 JobTracker 和 TaskTacker 进行分离，它由下面几大构成组件：

a. 一个全局的资源管理器 ResourceManager

b.ResourceManager 的每个节点代理 NodeManager

c. 表示每个应用的 ApplicationMaster

d. 每一个 ApplicationMaster 拥有多个 Container 在 NodeManager 上运行

YARN 架构简析

集中式架构

集中式调度器(Monolithic Scheduler)的特点是，资源的调度和应用程序的管理功能全部放到一个进程中完成，开源界典型的代表是 MRv1 JobTracker 的实现。这样设计的缺点很明显，扩展性差：首先，集群规模受限；其次，新的调度策略难以融入到现有代码中，比如之前仅支持 MapReduce 作业，现在要支持流式作业，而将流式作业的调度策略嵌入到中央调度其中是一项很难的工作。

双层调度架构

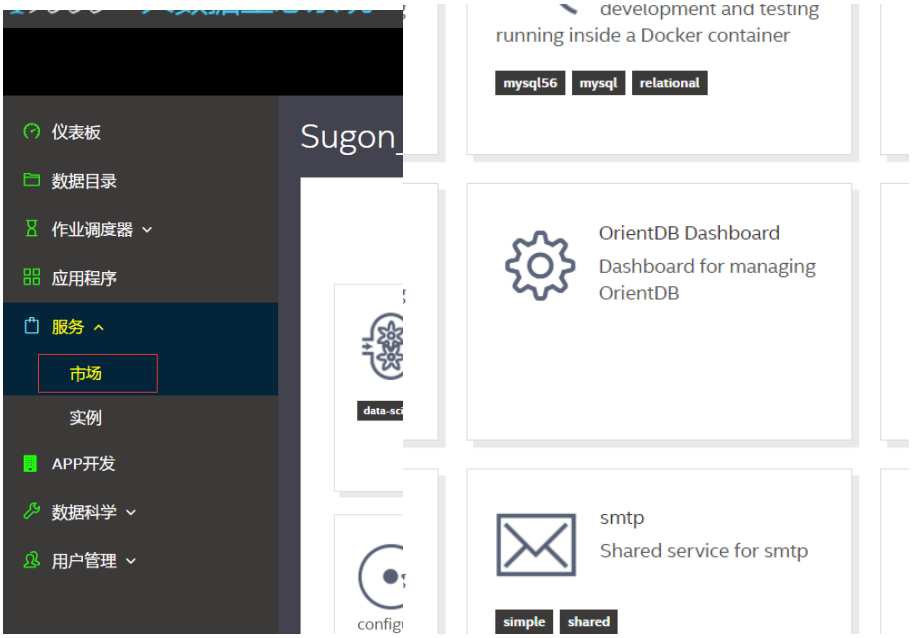
为了克服集中式调度器的不足，双层调度器是一种很容易被想到的解决之道，它可看作是一种分而治之的机制或者是策略下放机制 :双层调度器仍保留一个经简化的集中式资源调度器，但具体任务相关的调度策略则下放到各个应用程序调度器完成。这种调度器的典型代表是 Mesos。Mesos 调度器由两部分组成，分别是资源调度器和框架(应用程序)调度器,其中，资源调度器负责将集群中的资源分配给各个框架(应用程序),而框架(应用程序)调度器负责将资源进一步分配给内部的各个任务，用户很容易将一种框架或者系统接入 Mesos.

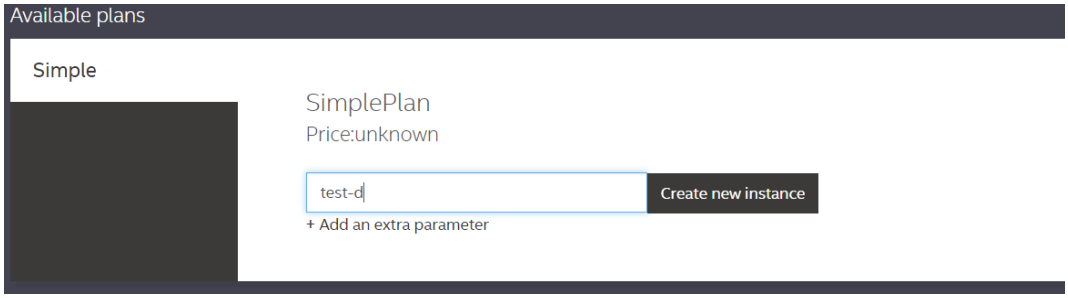
双层调度器的特点是：各个框架调度器并不知道整个集群资源使用情况，只是被动地接受资源；资源调度器仅将可用的资源推送给各个框架，而由框架自己选择是使用还是拒绝这

些资源；一旦框架接受到新资源，再进一步将资源分配给其内部的任务，进而实现双层调度。然而这种调度器也是有缺点，主要表现在以下两个方面:1.各个框架无法知道整个集群的实时资源使用情况；采用悲观锁，并发粒度小。

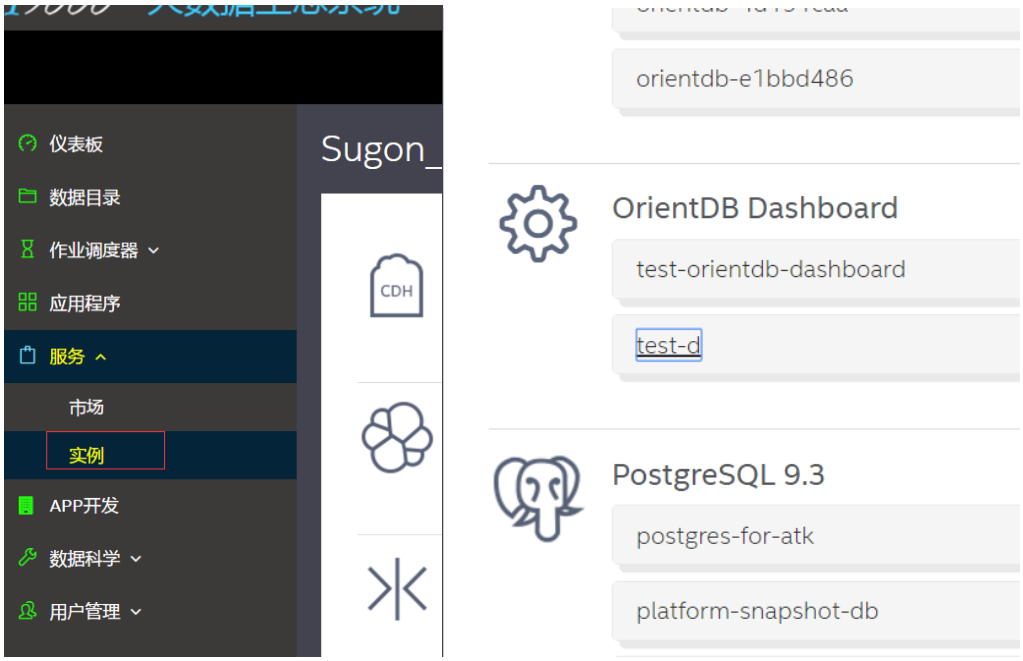
使用方法：

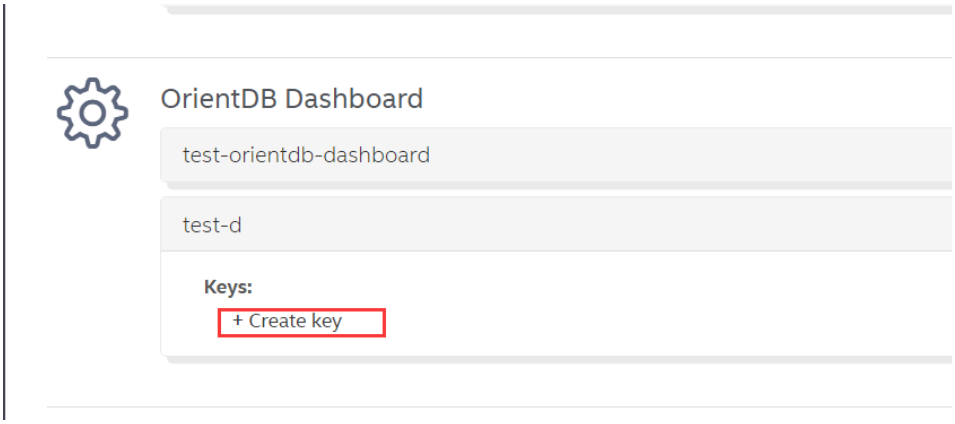
1. 在 i9000 平台上选择 服务>市场，找到 Yarn，点击 Yarn，创建一个新的实例。



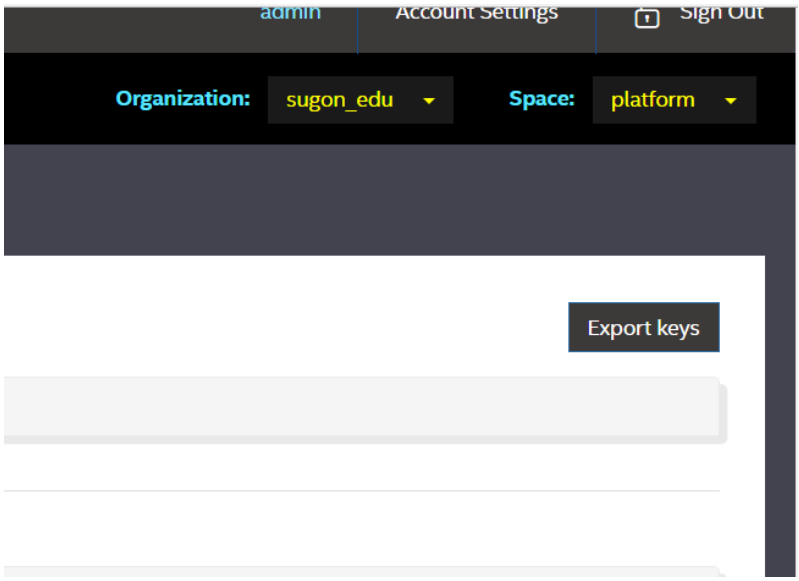


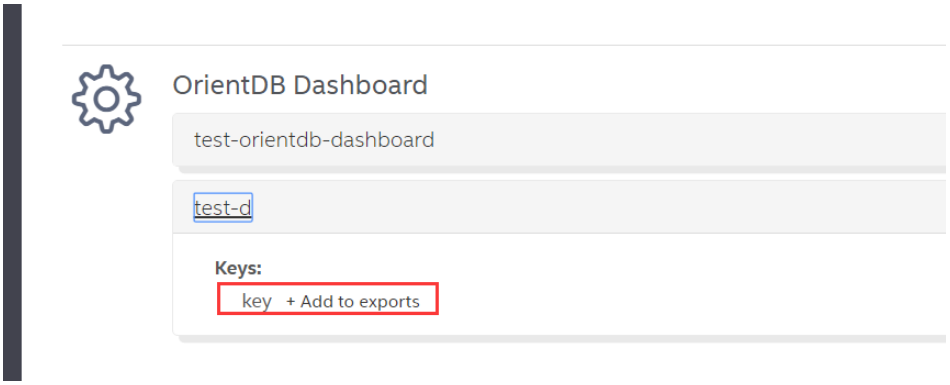
2. 在 服务>实例 里面找到 Yarn，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Yarn 的连接信息。





exported Keys:


```
{
  "yarn": [
    {
      "label": "yarn",
      "name": "test-yarn",
      "plan": "bare",
      "tags": [],
      "credentials": {
        "HADOOP_CONFIG_ZIP": {
          "description": "This is the encoded zip file of hadoop-configuration",
          "encoded_zip": "UEsDBBQACAgIAGU7LkoAAAAAAAAAAAAAAAAYAAAAeWFybi1jb25mL3NzbC1jbGllbnQueG1slY8xDsIwDEX3niJ0T1o2hjQVQmJjgwOE1lSB1KkcB9HbE2BBCAY8/S89vS/r9jZ6cQWKLmBTllVdCsAu9A6Mpjzst3JVtqYo9ELKdeIwAAJZhl4cZ7HxIfW5ip1FOwBJaQrdBTy5IWUoG00hhJ4oTEA8P0quaEcw4MXrVeQfIiilFjhwIFM8T6OoJvNir9QnM+RJ19YoPX/Uu/MNO4IPtlUMGyrYvQ8
```



```
        "yarn.resourcemanager.zk-address": "cdh-master-0.node.envname.consul:2181,cdh-master-1.node.envname.consul:2181,cdh-master-2.node.envname.consul:2181",
        "hadoop.security.groups.cache.secs": "1",
        "hadoop.proxyuser.mapred.hosts": "*",
        "yarn.application.classpath": "$HADOOP_CLIENT_CONF_DIR,$HADOOP_CONF_DIR,$HADOOP_COMMON_HOME/*,$HADOOP_COMMON_HOME/lib/*,$HADOOP_HDFS_HOME/*,$HADOOP_HDFS_HOME/lib/*,$HADOOP_YARN_HOME/*,$HADOOP_YARN_HOME/lib/*",
        "dfs.replication": "3",
        "yarn.resourcemanager.scheduler.address.rm39": "cdh-master-0.node.envname.consul:8030",
        "yarn.admin.acl": "mapred,yarn",
        "hadoop.security.auth_to_local": "DEFAULT",
        "yarn.scheduler.increment-allocation-vcores": "1",
        "yarn.resourcemanager.webapp.https.address.rm41": "cdh-master-1.node.envname.consul:8090",
        "yarn.resourcemanager.scheduler.address.rm41": "cdh-master-1.node.envname.consul:8030",
        "mapreduce.shuffle.max.connections": "80",
        "yarn.resourcemanager.webapp.https.address.rm39": "cdh-master-0.node.envname.consul:8090",
        "mapreduce.job.reduces": "1",
        "yarn.scheduler.maximum-allocation-vcores": "32",
        "yarn.resourcemanager.cluster-id": "yarnRM",
        "hadoop.proxyuser.HTTP.hosts": "*",
        "hadoop.proxyuser.httpfs.hosts": "*",
        "dfs.namenode.https-address.nameservice1.namenode21": "cdh-master-0.node.envname.consul:50470",
        "dfs.namenode.http-address.nameservice1.namenode21": "cdh-master-0.node.envname.consul:50070",
        "dfs.namenode.https-address.nameservice1.namenode22": "cdh-master-1.node.envname.consul:50470",
        "dfs.namenode.http-address.nameservice1.namenode22": "cdh-master-1.node.envname.consul:50070",
        "dfs.namenode.acls.enabled": "true",
        "yarn.scheduler.increment-allocation-mb": "512",
        "mapreduce.map.output.compress.codec": "org.apache.hadoop.io.compress.SnappyCodec",
        "yarn.resourcemanager.nm.liveness-monitor.interval-ms": "1000",
        "hadoop.proxyuser.mapred.groups": "*",
```

4. 在代码里使用此配置文件进行配置，即可使用新建的 Yarn。

```
public class RemoteMapReduceService {
    public static String startJob() throws Exception {
        Job job = Job.getInstance();
        job.setJobName("xxxx");
        /**************************
         *......
         *在这里，和普通的MapReduce一样，设置各种需要的东西
         *......
         **************************/

        //下面为了远程提交添加设置：
        Configuration conf = job.getConfiguration();
        conf.set("mapreduce.framework.name", "yarn");
        conf.set("hbase.zookeeper.quorum", "MASTER:2181");
        conf.set("fs.default.name", "hdfs://MASTER:8020");
        conf.set("yarn.resourcemanager.resource-tracker.address", "MASTER:8031");
        conf.set("yarn.resourcemanager.address", "MASTER:8032");
        conf.set("yarn.resourcemanager.scheduler.address", "MASTER:8030");
        conf.set("yarn.resourcemanager.admin.address", "MASTER:8033");
        conf.set("yarn.application.classpath", "$HADOOP_CONF_DIR,"
            +"$HADOOP_COMMON_HOME/*,$HADOOP_COMMON_HOME/lib/*,"
            +"$HADOOP_HDFS_HOME/*,$HADOOP_HDFS_HOME/lib/*,"
            +"$HADOOP_MAPRED_HOME/*,$HADOOP_MAPRED_HOME/lib/*,"
            +"$YARN_HOME/*,$YARN_HOME/lib/*,"
            +"$HBASE_HOME/*,$HBASE_HOME/lib/*,$HBASE_HOME/conf/*");
        conf.set("mapreduce.jobhistory.address", "MASTER:10020");
        conf.set("mapreduce.jobhistory.webapp.address", "MASTER:19888");
        conf.set("mapred.child.java.opts", "-Xmx1024m");

        job.submit();
        //提交以后，可以拿到JobID。根据这个JobID可以打开网页查看执行进度。
        return job.getJobID().toString();
    }
}
```

Zookeeper

概述：

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

ZooKeeper 的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

ZooKeeper 包含一个简单的原语集，提供 Java 和 C 的接口。

ZooKeeper 代码版本中，提供了分布式独享锁、选举、队列的接口，代码在 zookeeper-3.4.3\src\recipes。其中分布锁和队列有 Java 和 C 两个版本，选举只有 Java 版本。

工作原理：

Zookeeper 的核心是原子广播，这个机制保证了各个 Server 之间的同步。实现这个机制的协议叫做 Zab 协议。Zab 协议有两种模式，它们分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，Zab 就进入了恢复模式，当领导者被选举出来，且大多数 Server 完成了和 leader 的状态同步以后，恢复模式就结束了。状态同步保证了 leader 和 Server 具有相同的系统状态。

为了保证事务的顺序一致性，zookeeper 采用了递增的事务 id 号（zxid）来标识事务。所有的提议（proposal）都在被提出的时候加上了 zxid。实现中 zxid 是一个 64 位的数字，它高 32 位是 epoch 用来标识 leader 关系是否改变，每次一个 leader 被选出来，它都会

有一个新的 epoch，标识当前属于那个 leader 的统治时期。低 32 位用于递增计数。

每个 Server 在工作过程中有三种状态：

LOOKING：当前 Server 不知道 leader 是谁，正在搜寻

LEADING：当前 Server 即为选举出来的 leader

FOLLOWING：leader 已经选举出来，当前 Server 与之同步

特点：

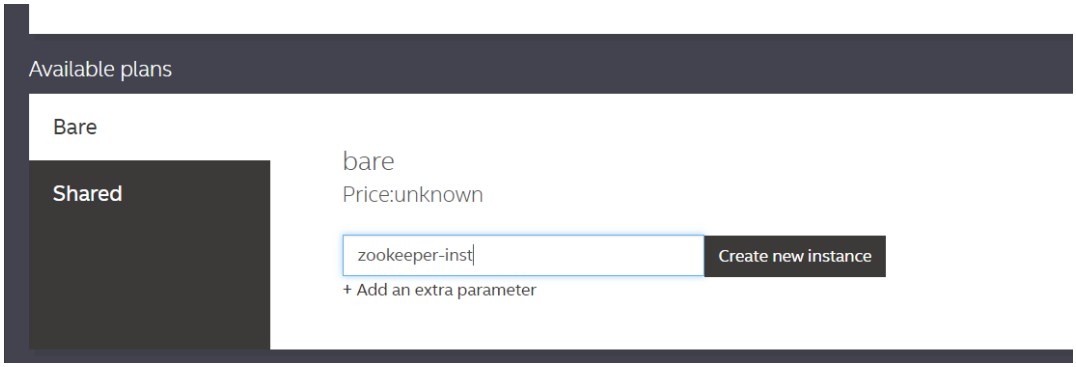
在 Zookeeper 中，znode 是一个跟 Unix 文件系统路径相似的节点，可以往这个节点存储或获取数据。如果在创建 znode 时 Flag 设置为 EPHEMERAL ,那么当创建这个 znode 的节点和 Zookeeper 失去连接后，这个 znode 将不再存在在 Zookeeper 里，Zookeeper 使用 Watcher 察觉事件信息。当客户端接收到事件信息，比如连接超时、节点数据改变、子节点改变，可以调用相应的行为来处理数据。Zookeeper 的 Wiki 页面展示了如何使用 Zookeeper 来处理事件通知，队列，优先队列，锁，共享锁，可撤销的共享锁，两阶段提交。

那么 Zookeeper 能做什么事情呢 ,简单的例子 :假设我们有 20 个搜索引擎的服务器(每个负责总索引中的一部分的搜索任务)和一个总服务器(负责向这 20 个搜索引擎的服务器发出搜索请求并合并结果集)，一个备用的总服务器(负责当总服务器宕机时替换总服务器)，一个 web 的 cgi(向总服务器发出搜索请求)。搜索引擎的服务器中的 15 个服务器提供搜索服务，5 个服务器正在生成索引。这 20 个搜索引擎的服务器经常要让正在提供搜索服务的服务器停止提供服务开始生成索引 ,或生成索引的服务器已经把索引生成完成可以提供搜索服务了。使用 Zookeeper 可以保证总服务器自动感知有多少提供搜索引擎的服务器并向这些

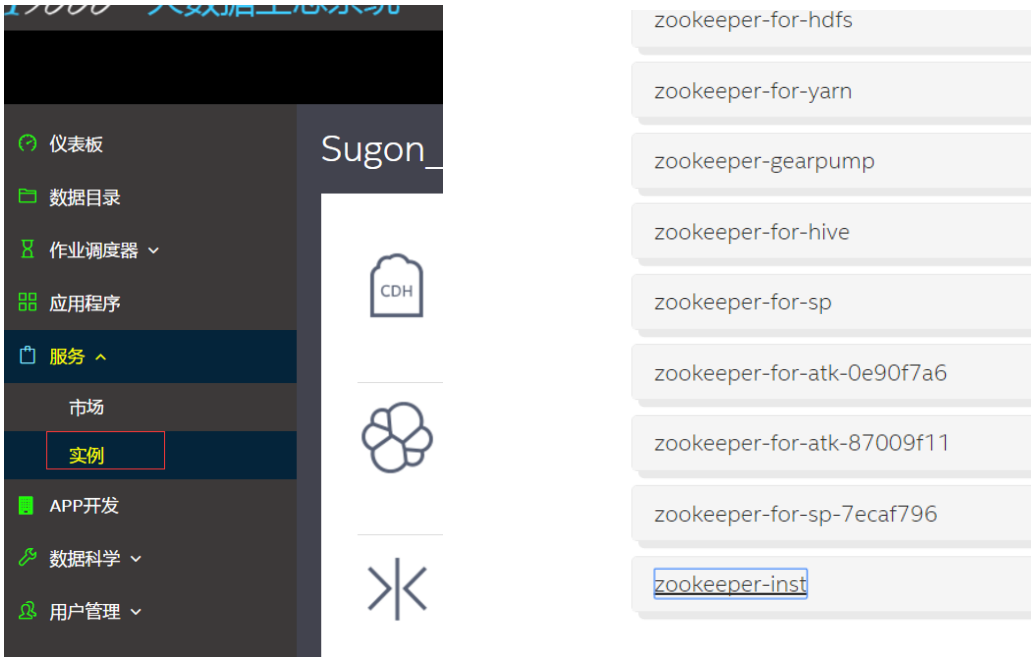
服务器发出搜索请求，当总服务器宕机时自动启用备用的总服务器。

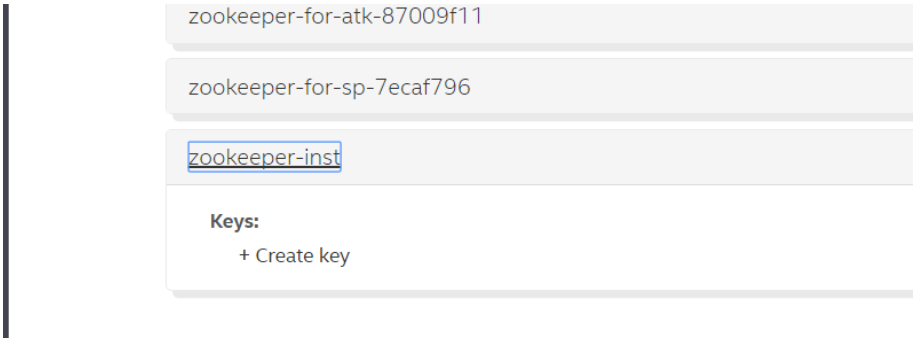
使用方法：

1. 在 i9000 平台上选择 服务>市场，找到 Zookeeper，点击 Zookeeper，创建一个新的实例。

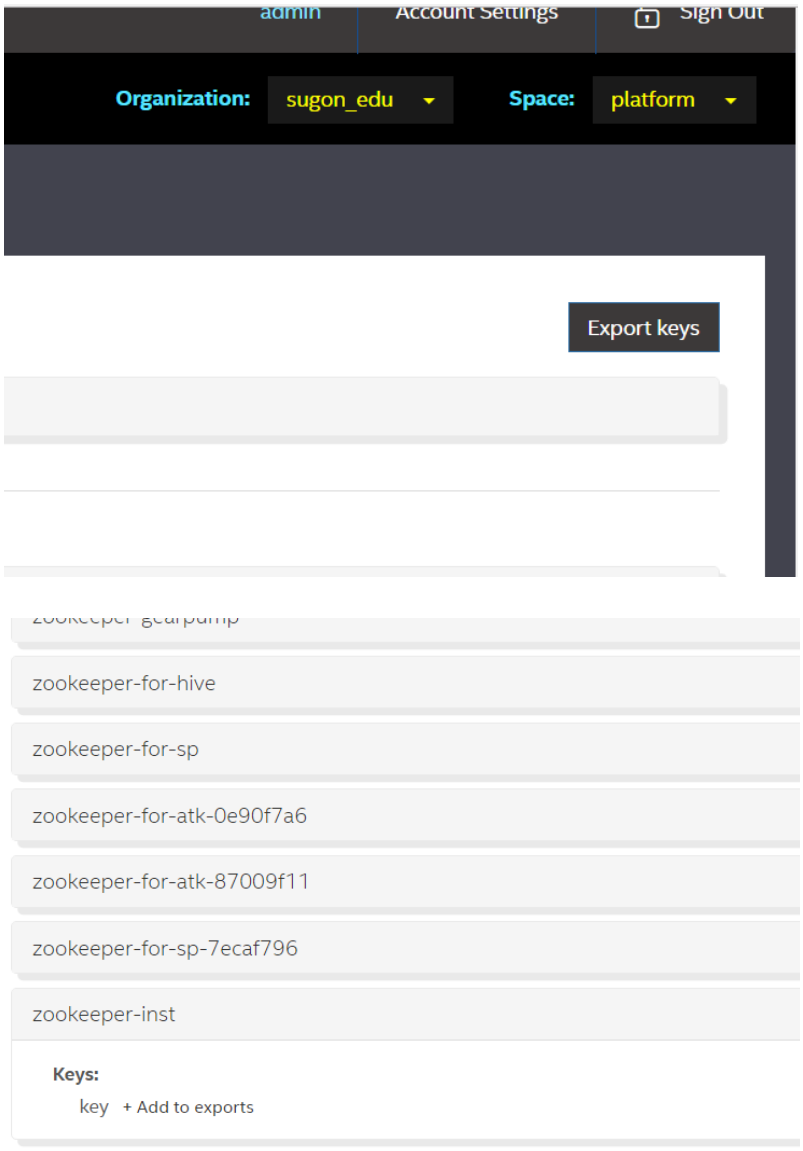


2. 在 服务>实例 里面找到 Zookeeper，找到刚刚创建的实例，点击，创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Zookeeper 的连接信息。



4. 在代码里使用 zk.cluster 即可连接到 Zookeeper。

```
*/
public class JavaApiSample implements Watcher {

    private static final int SESSION_TIMEOUT = 10000;
    private static final String CONNECTION_STRING = "test.zookeeper.connection_string:2181";
    private static final String ZK_PATH = "/nileader";
    private ZooKeeper zk = null;

    private CountDownLatch connectedSemaphore = new CountDownLatch( 1 );

    /**
     * 创建ZK连接
     * @param connectString  ZK服务器地址列表
     * @param sessionTimeout   Session超时时间
     */
    public void createConnection( String connectString, int sessionTimeout ) {
        this.releaseConnection();
        try {
            zk = new ZooKeeper( connectString, sessionTimeout, this );
            connectedSemaphore.await();
        } catch ( InterruptedException e ) {
            System.out.println( "连接创建失败，发生 InterruptedException" );
            e.printStackTrace();
        } catch ( IOException e ) {
            System.out.println( "连接创建失败，发生 IOException" );
            e.printStackTrace();
```

也可以在其他应用中绑定此实例。

```
services:
  - zookeeper-instance
```

Zookeeper-wssb

概述：

Zookeeper 是为分布式应用程序提供高性能协调服务的工具集合，也是 Google 的 Chubby 一个开源的实现，是 Hadoop 的分布式协调服务。它包含一个简单的原语集，分布式应用程序可以基于它实现配置维护、命名服务、分布式同步、组服务等。Zookeeper 可以用来保证数据在 ZK 集群之间的数据的事务性一致。其中 ZooKeeper 提供通用的分布式锁服务，用以协调分布式应用。

Zookeeper 作为 Hadoop 项目中的一个子项目,是 Hadoop 集群管理的一个必不可少的模块,它主要用来解决分布式应用中经常遇到的数据管理问题，如集群管理、统一命名服务、分布式配置管理、分布式消息队列、分布式锁、分布式协调等。在 Hadoop 中，它管理 Hadoop 集群中的 NameNode，还有在 Hbase 中 Master Election、Server 之间状态同状步等。

Zoopkeeper 提供了一套很好的分布式集群管理的机制，就是它这种基于层次型的目录树的数据结构，并对树中的节点进行有效管理，从而可以设计出多种多样的分布式的数据管理模型。

设计目标：

众所周知，分布式环境下的程序和活动为了达到协调一致目的，通常具有某些共同的特点，例如，简单性、有序性等。ZooKeeper 不但在这些目标的实现上有自身特点，并且具有独特优势。下面我们将简述 ZooKeeper 的设计目标。

（1）简单化

ZooKeeper 允许各分布式进程通过一个共享的命名空间相互联系，该命名空间类似于一个标准的层次型的文件系统：由若干注册了的数据节点构成(用 Zookeeper 的术语叫 znode)，这些节点类似于文件和目录。典型的文件系统是基于存储设备的，文传统的文件系统主要用于存储功能，然而 ZooKepper 的数据是保存在内存中的。也就是说，可以获得高吞吐和低延迟。ZooKeeper 的实现非常重视高性能、高可靠，以及严格的有序访问。

高性能保证了 ZooKeeper 可以用于大型的分布式系统，高可靠保证了 ZooKeeper 不会发生单点故障，严格的顺序访问保证了客户端可以获得复杂的同步操作原语。

（2）健壮性

就像 ZooKeeper 需要协调的分布式系统一样，它本身就是具有冗余结构，它构建在一系列主机之上，叫做一个”ensemble”。

构成 ZooKeeper 服务的各服务器之间必须相互知道，它们维护着一个状态信息的内存映像，以及在持久化存储中维护着事务日志和快照。只要大部分服务器正常工作，ZooKeeper 服务就能正常工作。

客户端连接到一台 ZooKeeper 服务器。客户端维护这个 TCP 连接，通过这个连接，客户端可以发送请求、得到应答，得到监视事件以及发送心跳。如果这个连接断了，客户端可以连接到另一个 ZooKeeper 服务器。

（3）有序性

ZooKeeper 给每次更新附加一个数字标签，表明 ZooKeeper 中的事务顺序，后续操作可以利用这个顺序来完成更高层次的抽象功能，例如同步原语。

（4）速度优势

ZooKeeper 特别适合于以读为主要负荷的场合。ZooKeeper 可以运行在数千台机器上，如果大部分操作为读，例如读写比例为 10:1，ZooKeeper 的效率会很高。

ZooKeeper 的特点：

（1）ZooKeeper 的读写机制

Zookeeper 是一个由多个 server 组成的集群

一个 leader，多个 follower

每个 server 保存一份数据副本

全局数据一致

分布式读写

更新请求转发，由 leader 实施

（2）ZooKeeper 的保证

ZooKeeper 运行非常快而且简单。虽然它的目标是构建更加复杂服务（例如同步）的基础，但它提供了一些保证，如下：

1.顺序一致性：来自于客户端的更新，根据发送的先后被顺序实施。

2.唯一的系统映像：尽管客户端连接到不同的服务器，但它们看到的一个唯一（一致性）的系统服务，client 无论连接到哪个 server，数据视图都是一致的。

3.可靠性：一旦实施了一个更新，就会一直保持那种状态，直到客户端再次更新它，同时数据更新原子性，一次数据更新要么成功，要么失败。

4.及时性：在一个确定的时间内，客户端看到的系统状态是最新的。

（3）ZooKeeper 特点

最终一致性 :client 不论连接到哪个 Server ,展示给它都是同一个视图 ,这是 zookeeper 最重要的性能。

可靠性：具有简单、健壮、良好的性能，如果消息 m 被一台服务器接受，那么它将被所有的服务器接受。

实时性：Zookeeper 保证客户端将在一个时间间隔范围内获得服务器的更新信息，或者服务器失效的信息。 但由于网络延时等原因，Zookeeper 不能保证两个客户端能同时得到刚更新的数据，如果需要最新数据，应该在读数据之前调用 sync()接口。

等待无关(wait-free)：慢的或者失效的 client，不得干预快速的 client 的请求，使得每个 client 都能有效的等待。

原子性：更新只能成功或者失败，没有中间状态。

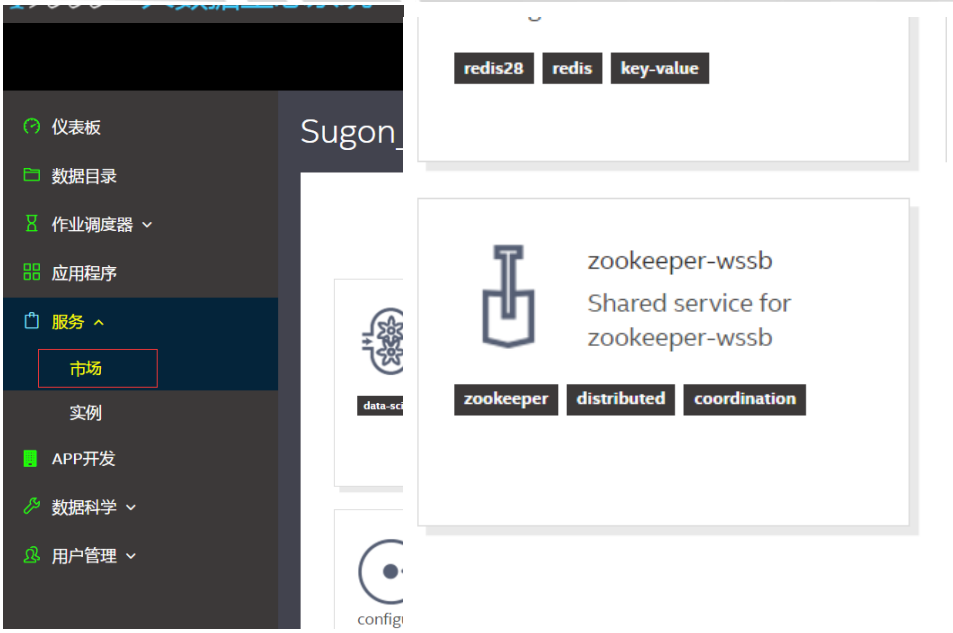
顺序性：包括全局有序和偏序两种：

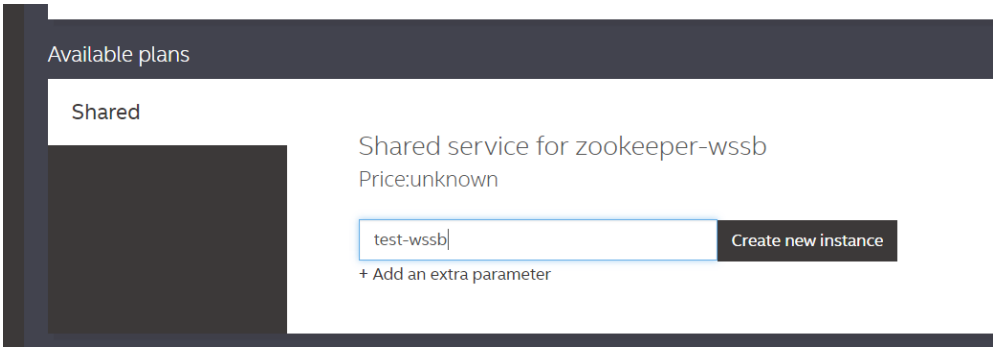
全局有序：是指如果在一台服务器上消息 a 在消息 b 前发布，则在所有 Server 上消息 a 都将在消息 b 前被发布；

偏序：是指如果一个消息 b 在消息 a 后被同一个发送者发布，a 必将排在 b 前面

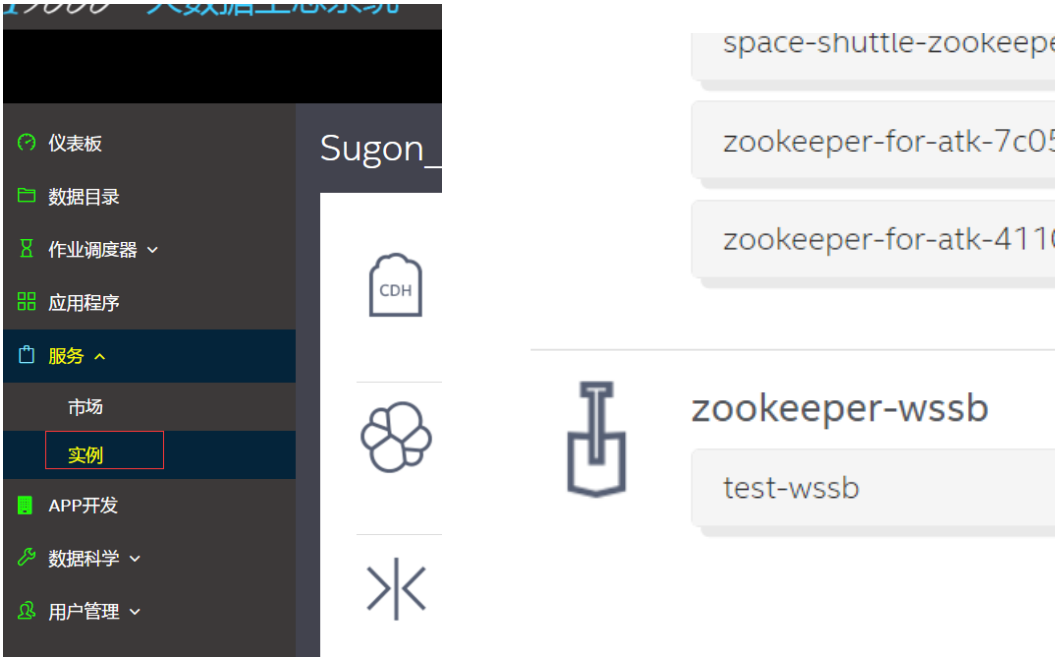
使用方法：

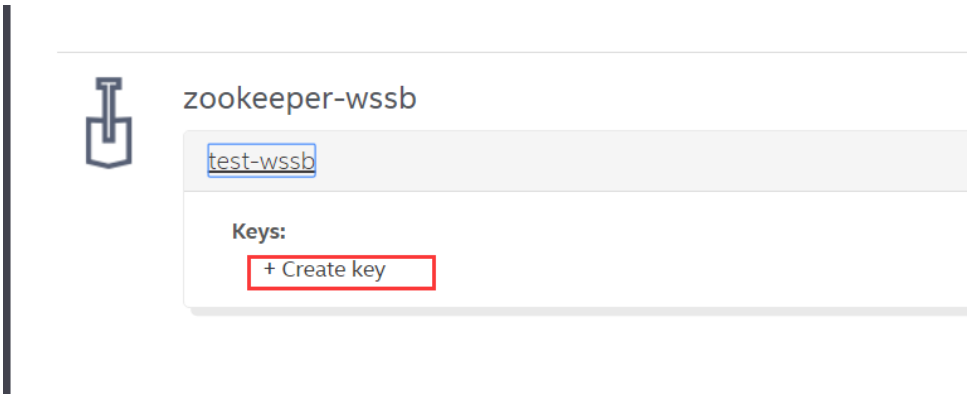
1. 在 i9000 平台上选择 服务>市场，找到 Zookeeper-wssb，点击 Zookeeper-wssb，创建一个新的实例。



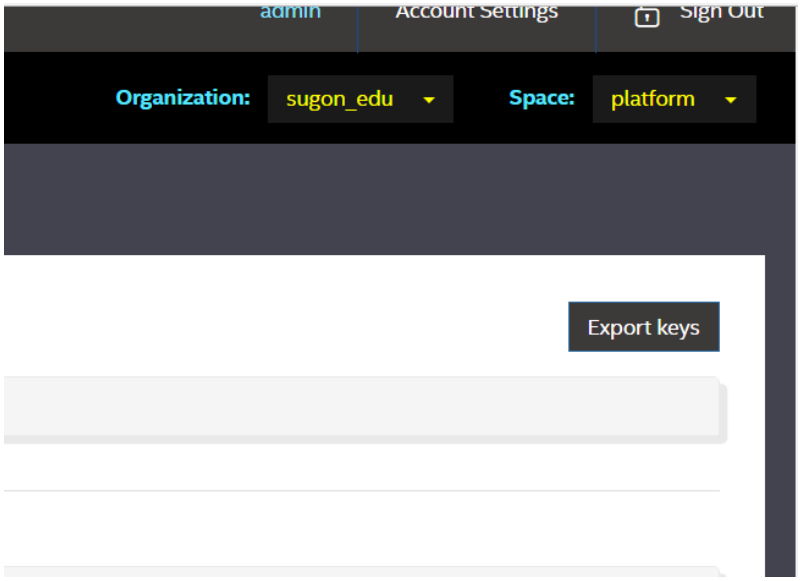


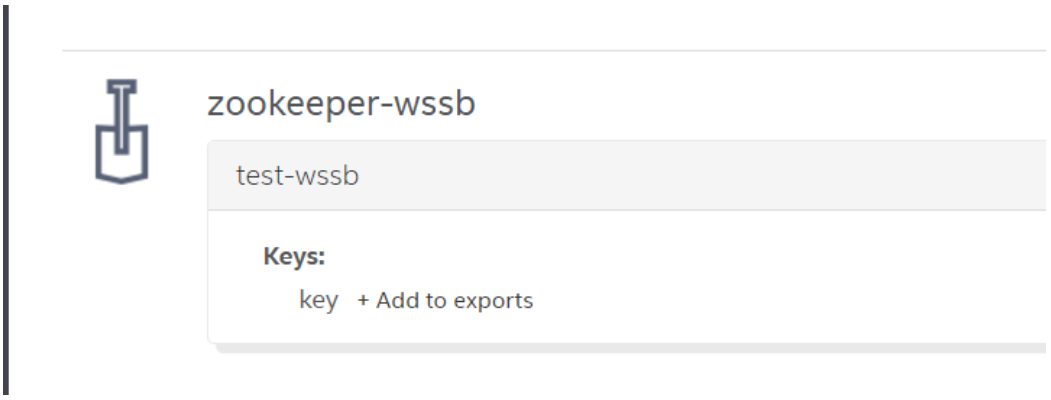
2. 在 服务>实例 里面找到 Zookeeper-wssb ,找到刚刚创建的实例 ,点击 ,创建新的 key。





3. 在页面右上角找到 Export keys 并点击，找到刚刚创建的 key，点 Add to exports，在页面最下面可以看到 Zookeeper-wssb 的连接信息。





exported Keys:


```
{
  "zookeeper-wssb": [
    {
      "label": "zookeeper-wssb",
      "name": "test-wssb",
      "plan": "shared",
      "tags": [
        "zookeeper",
        "distributed",
        "coordination"
      ],
      "credentials": {
        "uri": "cdh-master-0.node.envname.consul:2181,cdh-master-1.node.envname.consul:2181,cdh-master-2.node.envname.consul:2181"
      }
    }
  ]
}
```


4. 可在其他应用中绑定此实例。

```
---
applications:
- name: installer-rule-engine
  memory: 2048M
  disk_quota: 4095M
  timeout: 180
  instances: 1
services:
  - mycdh
  - myzookeeper-wssb
  - rule-engine-credentials-ups
  - dashboard-endpoint-ups
  - installer-backend-ups
env:
  VERSION: "0.15.0"
```

曙光瑞翼教育合作中心

地址：北京市海淀区万柳亿城中心C2座1104室
电话：010-58815892