Openstack+Docker

房秋生，2016-1-19

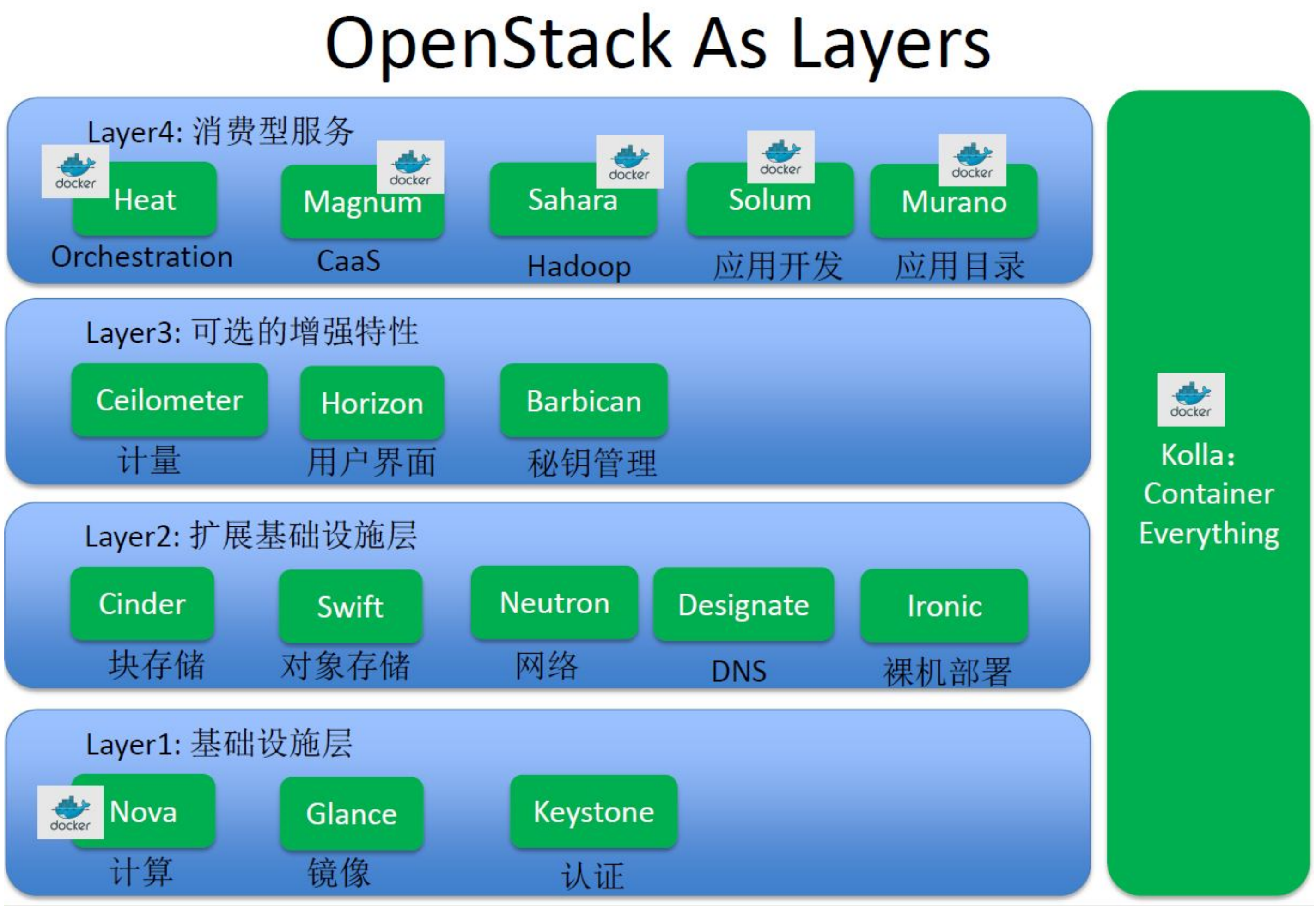
OpenStack-related Components

1. **openstack/python-magnumclient:** Python client for Container Infrastructure Management Service
2. **openstack/magnum:** Container Infrastructure Management Service for OpenStack
3. **openstack/magnum-ui:** Container Infrastructure Management Service for OpenStack
4. **openstack/fuxi:** Enable Docker container to use Cinder volume and Manila share
5. **openstack/higgins:** Container Management Service for OpenStack
6. **openstack/kolla:** Kolla provides production-ready containers and deployment tools for operating OpenStack clouds
7. **openstack/kolla-kubernetes:** Kubernetes deployment of the Kolla containers
8. **openstack/kuryr:** Bridge between container framework networking and storage models to OpenStack networking and storage abstractions.
9. ***Nova-docker:*** Nova Docker Driver

Docker vs OpenStack

1. **Docker:** PaaS Application Centric
2. **OpenStack:** IaaS Resource Centric

OpenStack Layers



Nova Docker Driver

Docker+Nova,Wikipage

1.Overview

The Docker driver is a hypervisor driver for Openstack Nova Compute. It was introduced with the Havana release, but lives out-of-tree for Icehouse and Juno. Being out-of-tree has allowed the driver to reach maturity and feature-parity faster than would be possible should it have remained in-tree. It is expected the driver will return to mainline Nova in the Kilo release.

Docker is an open-source engine which automates the deployment of applications as highly portable, self-sufficient containers which are independent of hardware, language, framework, packaging system and hosting provider.

Docker provides management of Linux containers with a high level API providing a lightweight solution that runs processes in isolation. It provides a way to automate software deployment in a secure and repeatable environment. A Docker container includes a software component along with all of its dependencies - binaries, libraries, configuration files, scripts, virtualenvs, jars, gems, tarballs, etc. Docker can be run on any x64 Linux kernel supporting cgroups and aufs.

Docker is a way of managing multiple containers on a single machine. However used behind Nova makes it much more powerful since it’s then possible to manage several hosts, which in turn manage hundreds of containers. The current Docker project aims for full OpenStack compatibility.

Containers don't aim to be a replacement for VMs, they are complementary in the sense that they are better for specific use cases.

2.What unique advantages Docker bring over other containers technologies?

Docker takes advantage of containers and filesystem technologies in a high-level which are not generic enough to be managed by libvirt.

Process-level API: Docker can collect the standard outputs and inputs of the process running in each container for logging or direct interaction, it allows blocking on a container until it exits, setting its environment, and other process-oriented primitives which don’t fit well in libvirt’s abstraction.

Advanced change control at the filesystem level: Every change made on the filesystem is managed through a set of layers which can be snapshotted, rolled back, diff-ed etc.

Image portability: The state of any docker container can be optionally committed as an image and shared through a central image registry. Docker images are designed to be portable across infrastructures, so they are a great building block for hybrid cloud scenarios.

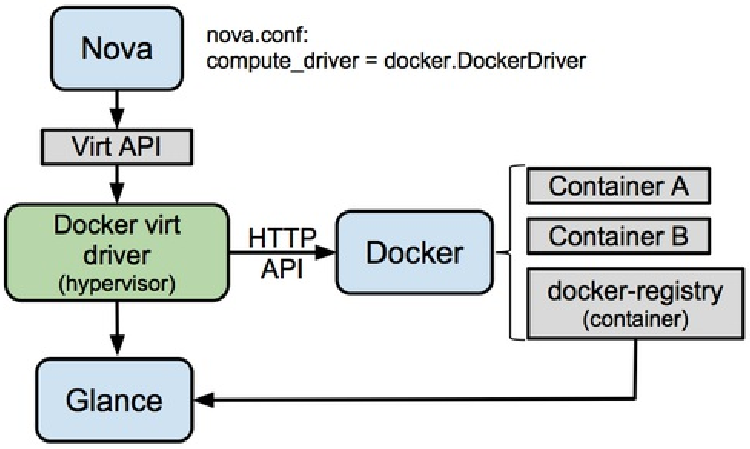
Build facility: docker can automate the assembly of a container from an application’s source code. This gives developers an easy way to deploy payloads to an OpenStack cluster as part of their development workflow.

3.How does the Nova hypervisor work under the hood?

The Nova driver embeds a tiny HTTP client which talks with the Docker internal Rest API through a unix socket. It uses the HTTP API to control containers and fetch information about them.

The driver will fetch images from the OpenStack Image Service (Glance) and load them into the Docker filesystem. Images may be placed in Glance by exporting them from Docker using the 'docker save' command.

Older versions of this driver required running a private docker-registry, which would proxy to Glance. This is no longer required.


Nova
nova.conf:
compute_driver = docker.DockerDriver
Virt API
Docker virt driver (hypervisor)
HTTP API
Docker
Container A
Container B
docker-registry (container)
Glance

4.Configure an existing OpenStack installation to enable Docker

Installing Docker for OpenStack:See Details in Docker+Nova,Wikipage

5.Summary

Nova Docker Driver把Docker作为一种新的Hypervisor来处理，作为一个nova compute的driver，便于集成。

优点：

1. 通过nova scheduler来进行资源调度；
2. 通过Heat来管理部署运行，服务发现和扩容缩容，所有的dockercontainer作为VM来处理
3. 通过Neutron来管理网络，GRE，vLan，VxLan等等，实现网络隔离。
4. 支持多租户，为不同租户设置不同的quota

缺点：丢失Docker的高级的特性

1. 容器关联
2. 端口映射
3. 不同网络模式的配置：•Host •Container

HEAT Docker Driver

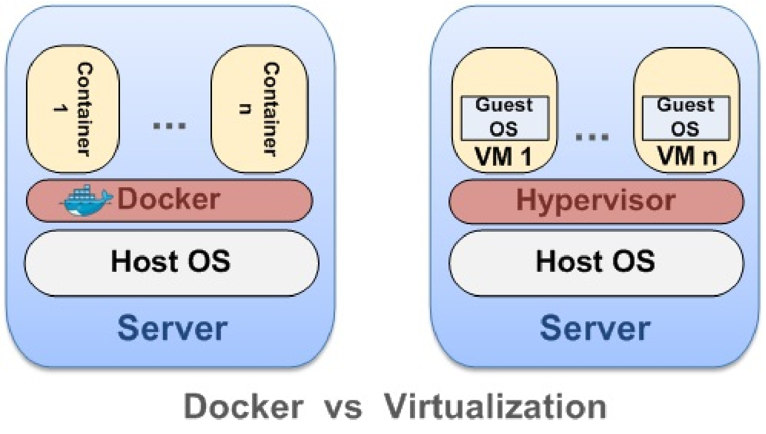
Cited Web: Docker-containers-deployment-with-OpenStack-Heat

1.What Is Docker?

Docker is an open source project to automatically deploy applications into containers. It commoditizes the well known LXC (Linux Container) solution that provides operating system level virtualization and allows to run multiple containers on the same server.

To make a simple analogy, a Docker is like an hypervisor. But unlike traditional Vms, docker containers are lightweight as they don’t run OSes but share the host’s operating system (see the figure below).

A docker container is also portable, it hosts the application and its dependencies and it is able to be deployed or relocated on any Linux server.



The Docker element that manages containers and deploys applications on them is called Docker Engine.

The second component of Docker is Docker Hub. We found it really Awesome ! It's the Docker's repository of application. You can share your application with your team members and you can ship and run it anywhere... It's really amazing :)

It was a quick introduction to Docker ! Now, let's see how to use it with OpenStack ;)

2.OpenStack and Docker

Openstack can be easily enhanced by docker plugins. Docker can be integrated into OpenStack Nova as a form of hypervisor (Containers used as VMs). But there is a better way to use Docker with OpenStack. It to orchestrate containers with OpenStack Heat !

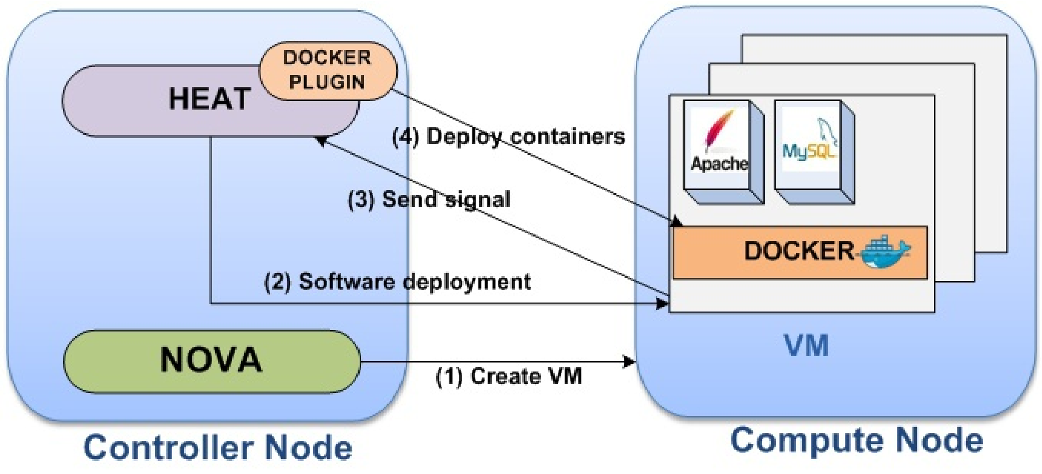
You have just to install the docker plugin on Heat and you will be able to create containers and deploy your applications on the top of them. You need to identify the required resources, edit your template and deploy it on Heat. Your stack will be created! (For detailed information on Heat and template creation, you can see our "Heat Usage Guide".

If you want to test Docker with Heat, we recommand you to deploy OpenStack using a flat networking model (see ""Our Guide"". One issue we encountered when we considered a multi-node architecture with isolated neutron networks is that instances were not able to signal to Heat. So, stack creation fails because it depends on connectivity from VMs to the Heat engine.

The figure below shows the communication between Heat, Nova, Docker and the instance when creating a stack. In this example we have deployed apache and mysql on two Docker containers. Stack deployment fails if the signal (3) can not reach Heat API.

I think this limitation will be overcome in the next versions to allow users using isolated neutron networks to deploy and test Docker!

But now you can enjoy docker on OpenStack with Flat-networking ;)



3.Deploy Docker containers with OpenStack Heat

3.1 Install the Docker Plugin

3.2 Create your Heat template

3.3 Deploy your stack

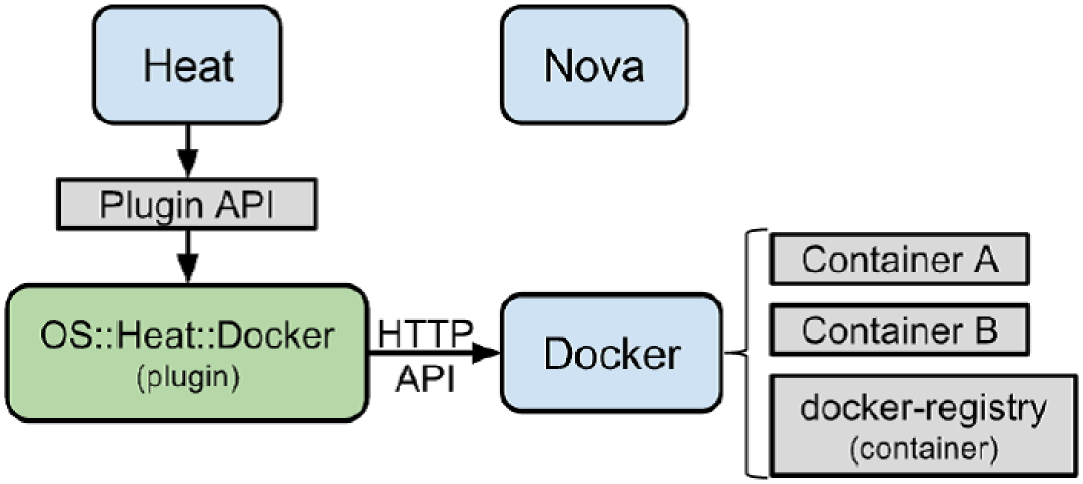
- Pre-deployment
- Create your stack

See details in Cited Web: Docker-containers-deployment-with-OpenStack-Heat

4.Summary

HEAT集成

- 添加了一个新的HEAT Resource：DockerInc::Docker::Container
- HEAT dockerdriver直接与dockerserver交互
- 没有和nova，cinder，neutron等交互



优点

- 完全兼容dockerAPI
- Docker所有参数可以在heat的template进行设置

缺点

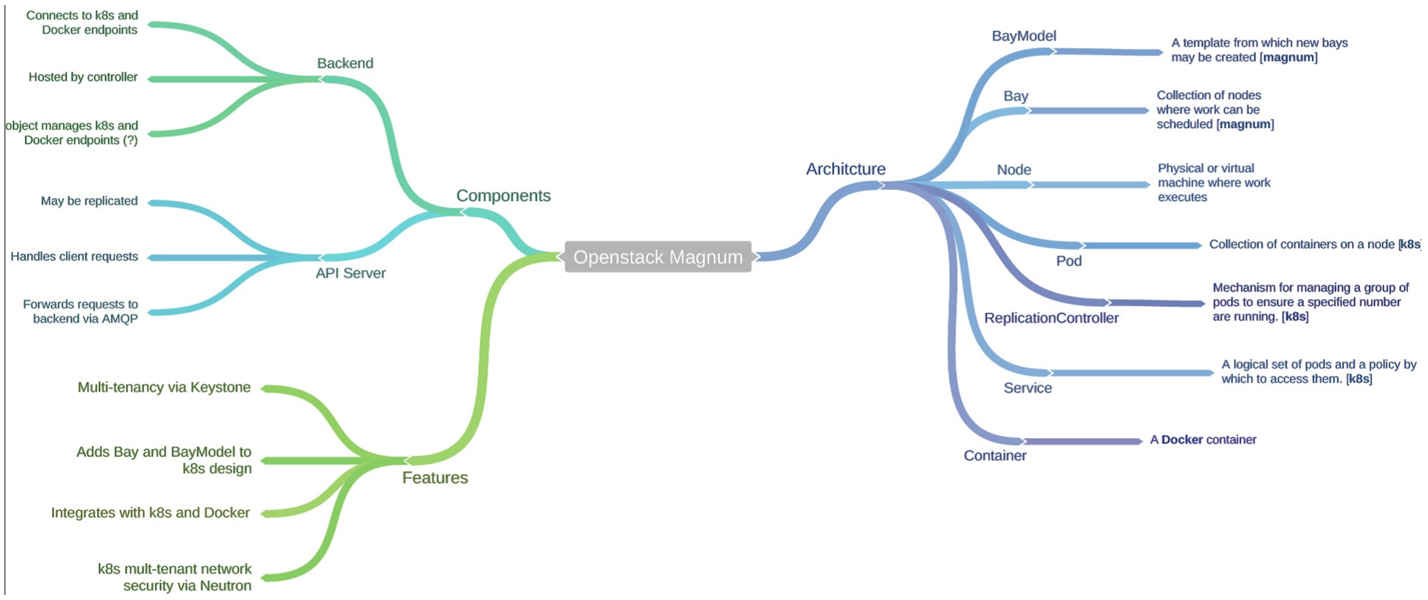
- 没有资源调度
- 没有网络管理

Magnum: Container As A Service

Containers Service

Containers share many features in common with Nova instances. For the common features, virt drivers for Nova can be used to surface basic instance functionality. For features that go beyond what can be naturally fit within a virt driver, we propose a new API service that allows for advanced features to be added without conflating the worlds of instances and containers.

Some examples of containers specific features are setting of shell environment variables, and accepting a shell command to execute at runtime. Capturing the STDIO of the process(es) within a container, and tracking the return status of processes are all beyond the scope of what was contemplated for Nova. All of these features will be implemented in the Containers Service.


Openstack Magnum
Components
Backend
Connects to k8s and Docker endpoints
Hosted by controller
object manages k8s and Docker endpoints (?)
API Server
May be replicated
Handles client requests
Forwards requests to backend via AMQP
Features
Multi-tenancy via Keystone
Adds Bay and BayModel to k8s design
Integrates with k8s and Docker
k8s mult-tenant network security via Neutron
Architcture
BayModel
A template from which new bays may be created [magnum]
Bay
Collection of nodes where work can be scheduled [magnum]
Node
Physical or virtual machine where work executes
Pod
Collection of containers on a node [k8s
ReplicationController
Mechanism for managing a group of pods to ensure a specified number are running. [k8s]
Service
A logical set of pods and a policy by which to access them. [k8s]
Container
A Docker container

Problem description

Container technology is rapidly gaining popularity as a way to bundle and deploy applications. Recognizing and adapting to this trend will position OpenStack to be useful not only to clouds that employ bare metal and virtual machine instances, but can remain competitive in offering container services as well.

Nova's concepts of an instance, and the actions that may be taken on it do not match completely with containers.

Use cases

1. App Consolidation. End-user wants to run multiple small applications in separate operating system environments, but wants to optimize for efficiency to control hosting costs. Each application belongs to the same tenant, so security isolation between applications is nice-to-have but not critical. Isolation is desired primarily for simplified management of the execution environment for each application.
2. App Portability. End-user wants to create a single container image, and deploy the same image to multiple hosting environments, including OpenStack. Other environments may include local servers, dedicated servers, private clouds, and public clouds. Switching environments requires passing database connection strings by environment variables at the time a container starts to allow the application to use the services available in each environment without changing the container image.
3. Docker Compatibility. End-user has a Dockerfile used to build an application and its runtime environment and dependencies in a Docker container image. They want an easy way to run the Docker resulting image on an OpenStack cloud.
4. LXC Compatibility. End-user wants an easy way to remotely create multiple LXC containers within a single Nova instance.
5. OpenVZ Compatibility. End-user wants an easy way to remotely create multiple OpenVZ containers within a single Nova instance.
6. Containers-Centric World View. End-user wants to communicate with a single OpenStack API, and request the addition of containers, without the need to be concerned with keeping track of how many containers are already running on a given Nova instance, and when more need to be created. They want to simply create and remove containers, and allow the appropriate resource scheduling to happen automatically.
7. Platform Integration. Cloud operator already has an OpenStack cloud, and wants to add a service/application centric management system on top. Examples of such systems are Cloud Foundry, Kubernetes, Apache Mesos, etc. The selected system is already Docker compatible. Allow this cloud operator easy integration with OpenStack to run applications in containers. The Cloud Operator now harnesses the power of both the management system, and OpenStack, and does not need to manage a second infrastructure for his/her application hosting needs. All details involving the integration of containers with Nova instances is managed by OpenStack.
8. Container network. End-user wants to define a custom overlay network for containers, and wants to have admin privilege to manage the network topology. Building a container network can decouple application deployment and management from the underlying network infrastructure, and enable additional usage scenario, such as (i) software-defined networking, and (ii) extending the container network (i.e. connecting various resources from multiple hosting environments). End-users want a single service that could help them build the container network, and dynamically modify the network topology by adding or removing containers to or from the network.
9. Permit secure use of native REST APIs. Provide two models of operation with Magnum. The first model allows Magnum to manage the lifecycle of Pods, ReplicationControllers, and Services. The second model allows end-users to manage the lifecycle of Pods, ReplicationControllers,

and Services by providing direct secure access to the native ReST APIs in Kubernetes and possibly Docker.

Long Term Use Cases

These use cases have been identified by the community as important, but unlikely to be tackled in short term (especially prior to incubation). We wish to adapt to these use cases in long term, but this is not a firm project commitment.

1. Multi-region/multi-cloud support. End-user wants to deploy applications to multiple regions/clouds, and dynamically relocate deployed applications across different regions/clouds. In particular, they want a single service that could help them (i) provision nodes from multiple regions/clouds, thus running containers on top of them, and (ii) dynamically relocate containers (e.g. through container migration) between nodes regardless of the underlying infrastructure.

Proposed change

Add a new API service for CRUD and advanced management of containers. If cloud operators only want to offer basic instance features for their containers, they may use nova with an alternate virt-driver, such as libvirt/lxc or nova-docker. For those wanting a full-featured container experience, they may offer the Containers Service API as well, in combination with Nova instances that contain an OpenStack agent that connects to the containers service through a security controlled agent (daemon) that allows the OpenStack control plane to provision and control containers running on Compute Hosts.

The Containers Service will call the Nova API to create one or more Nova instances inside which containers will be created. The Nova instances may be of any type, depending on the virt driver(s) chosen by the cloud operator. This includes bare-metal, virtual machines, containers, and potentially other instance types.

This allows the following configurations of containers in OpenStack.

- Containers in Virtual Machine Instances
- Containers in Bare Metal Instances
- Containers in Container Instances (nested)

The concept of nesting containers is currently possible if the parent container runs in privileged mode. Patches to the linux kernel are being developed to allow nesting of non-privileged containers as well, which provides a higher level of security.

The spirit of this plan aims to duplicate as little as possible between Nova and the Containers Service. Common components like the scheduler are expected to be abstracted into modules, such as Gantt that can be shared by multiple projects. Until Gantt is ready for use by the Containers Service, we will implement only two provisioning schemes for containers:

Create a container on a specified instance by using a nova instance guid. Auto-create instances (applies only until the Gantt scheduler is used) 2.1. Fill them sequentially until full. 2.2. Remove them automatically when they become empty. The above orchestration will be implemented using Heat. This requires some kind of hypervisor painting (such as host aggregates) for security reasons.

The diagram below offers an overview of the system architecture. The OSC box indicates an OpenStack client, which will communicate with the Containers Service through a REST API. The containers service may silently create Nova instances if one with enough capacity to host the requested container is not already known to the Containers service. The containers service will maintain a database "Map" of containers, and what Nova instance each belongs to. Nova creates instances. Instances are created in Nova, and containers belong only to the Containers Service, and run within a Nova instance. If the instance includes the agent software "A", then it may be included in the inventory of the Containers service. Instances that do not contain an agent may not interact with the Containers Service, and can be controlled only by a Nova virt driver.

Design Principles

1. Leverage existing OpenStack projects for what they are good at. Do not duplicate functionality, or copy code that can be otherwise accessed through API calls.
2. Keep modifications to Nova to a minimum.
3. Make the user experience for end users simple and familiar.
4. Allow for implementation of all features containers are intended to offer.

Alternatives

1. Extending Nova's existing feature set to offer container features 1.1. Container features don't fit into Nova's idea of compute (VM/Server)
2. A completely separate containers service forked from Nova. 2.1. Would result in large overlap and duplication in features and code

Data model impact

For Nova, None. All new data planned will be in the Containers Service.

REST API impact

For Nova, none. All new API calls will be implemented in the Containers Service. The OpenStack Containers Service API will be a superset of functionality offered by the, The Docker Remote API: with additionals to make is suitable for general use regardless of the backend container technology used, and to be compatible with OpenStack multi-tenancy and Keystone authentication.

Specific Additions:

- Support for the X-Auth-Project-Id HTTP request header to allow for multi-tenant use.
- Support for the X-Auth-Token HTTP request header to allow for authentication with keystone.

If either of the above headers are missing, a 401 Unauthorized response will be generated.

Docker CLI clients may communicate with a Swarmd instance that is configured to use the OpenStack Containers API as the backend for libswarm. This will allow for tool compatibility with the Docker ecosystem using the officially supported means for integration of a distributed system.

The scope of the full API will cause this spec to be too long to review, so the intent is to deal with the specific API design as a series of Gerrit reviews that submit API code as Not Implemented stubs with docstrings that clearly document the design, so allow for approval, and further implementation.

Security impact

Because Nova will not be changed, there should be no security impacts to Nova. The Containers Service implementation, will have the following security related issues:

1. Need to authenticate against keystone using python-keystoneclient.
2. A trust token from Nova will be needed in order for the Containers Service to call the Nova API on behalf of a user.
3. Limits must be implemented to control resource consumption in accordance with quotas.
4. Providing STDIO access may generate a considerable amount of network chatter between containers and clients through the relay. This could lead to bandwidth congestion at the relays, or API nodes. An approach similar to how we handle serial console access today will need to be considered to mitigate this concern.

Using containers implies a range of security considerations for cloud operators. These include:

- Containers in the same instance share an operating system. If the kernel is exploited using a security vulnerability, processes in once container may escape the constraints of the container and potentially access other resources on the host, including contents of other containers.
- Output of processes may be persisted by the containers service in order to allow asynchronous collection of exit status, and terminal output. Such content may include sensitive information.Features may be added to mitigate the risk of this data being replicated in log messages, including errors.
- Creating containers usually requires root access. This means that the Agent may need to be run with special privileges, or be given a method to escalate privileges using techniques such as sudo.
- User provided data is passed through the API. This will require sensible data input validation.

Notifications impact

Contemplated features (in subsequent release cycles):

- Notify the end user each time a Nova instance is created or deleted by the Containers service, if (s)he has registered for such notifications.

- Notify the user each on CRUD of containers containing start and end notifications. (compute.container.create/delete/etc)

- Notify user periodically of existence of container service managed containers (ex compute.container.exists)

Other end user impact

The user interface will be a REST API. On top of that API will be an implementation of the libswarm API to allow for tools designed to use Docker to treat OpenStack as an upstream system.

Performance Impact

The Nova API will be used to create instances as needed. If the Container to Instance ratio is 10, then the Nova API will be called at least once for every 10 calls to the Containers Service. Instances that are left empty will be automatically deleted, so in the example of a 10:1 ratio, the Nova API will be called to perform a delete for every 10 deletes in the Container Service. Depending on the configuration, the ratio may be as low as 1:1. The Containers Service will only access Nova through its API, not by accessing its database.

Other deployer impact

Deployers may want to adjust the default flavor used for Nova Instances created by the Containers Service.

There should be no impact on users of prior releases, as this introduces a new API.

Developer impact

Minimal. There will be minimal changes required in Nova, if any.

Magnum Roadmap

Cited Web:https://blueprints.launchpad.net/magnum

Magnum Conductor 水平扩展

- 提高系统处理client请求的性能
- 原生Docker集群调度管理
- Swarm, Gantt, Mesos

原生Docker集群网络管理

- 在Docker服务器上通过--net=host模式部署dockercontainer启动L2 Agent

Magnum Notifications

- 资源状态跟踪
- 监控
- 第三方集成

K8S深度集成

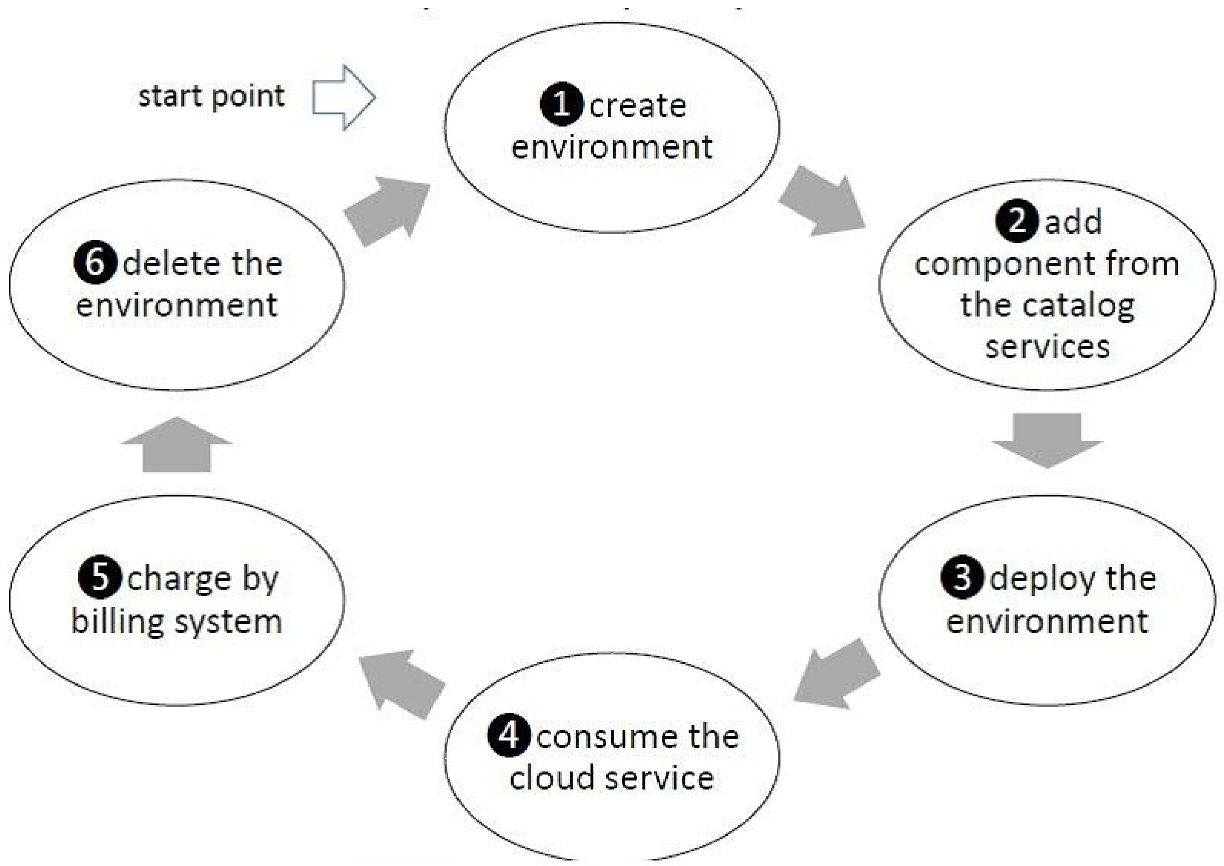
- Mangum提供python-k8sclient和K8S集成

Murano: Application Catalog

Murano简介

- 在OpenStack基础上提供应用目录服务
- 应用开发人员或者云管理员可以通过Solum开发应用并通过Murano发布这些应用
- 用户挑选自己需要的应用服务，通过应用服务组合构建自己的应用。
- Murano通过HEAT部署应用
- Murano通过Ceilometer触发自动扩展
- Kubernetes现在已经作为Murano的一个服务

Murano应用一键部署


start point
❶ create environment
❷ add component from the catalog services
❸ deploy the environment
❹ consume the cloud service
❺ charge by billing system
❻ delete the environment

- Create environment: 设置环境名，例如KubernetsHttpd
- Add component ...:1）创建K8S集群模板; 2）创建Pod模板; 3）创建Container模板
- Deploy the environment: 1）通过heat创建K8S集群; 2）通过K8S创建Pod

Kolla:Docker+OpenStack

Kolla简介

- 使用Docker容器部署OpenStack服务
- 提供所有OpenStack服务的镜像
- 基于docker-compose
- 所有的container采用--net=host模式启动

为什么需要Kolla

- 简化OpenStack的安装部署和回滚
- 按照组件/服务对OpenStack进行升级/回滚

创建一个OpenStack集群

- 启动一个管理节点 •$ ./kolla/tools/start
- 启动一个计算节点 •$ docker-compose -f nova-compute-network.ymlup –d
- 启动单个服务 –$ dockerrun --name glance-api-d --net=host --env-file=openstack.envkollaglue/fedora-rdo-glance-api:latest

如何选择?

Nova DockerDriver

- 将以前的VM workload迁移到DockerContainer
- 轻量的虚拟化技术

- 一个很典型的例子是Sahara通过heat调用nova dockerdriver创建hadoop集群

Heat DockerDriver

- 简单通过OpenStack测试Docker的一些高级功能
- 小规模Docker集群部署

Magnum

- 对K8S，CoreOS，Swarm不熟，有一定的OpenStack经验
- 大规模Docker集群管理，集群需要self-healing的能力
- 多租户
- 混合云部署Docker
- 使用Docker的一些高级功能，如“exec”，“link”等等

Murano

- 想通过K8S来管理Docker
- 混合云部署Docker

Kolla

- 简化OpenStack安装部署和升级

OpenStack +Docker: 资源管理改进

Docker& IaaS

- Docker的出现使得IaaS的计算密度进一步提升
- 密集型计算对资源管理和资源调度提出了跟高的要求

如何提高资源利用率？

层级的多租户管理

- Nova在Kilo版会加入最基本的层级多租户的支持
- 只有资源独占（最简单的资源规划）
- 不同租户之间不能共享资源

Kilo的多租户模式

...

多租户借入/借出模式

...

多租户共享模式

...