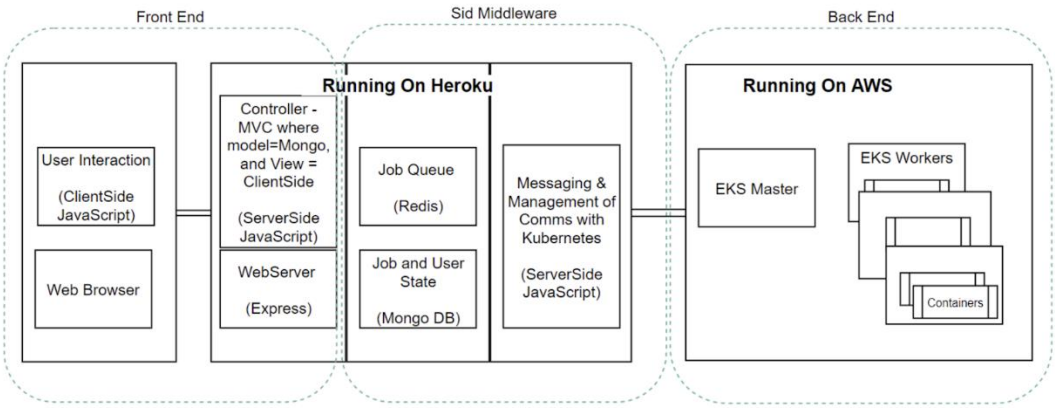
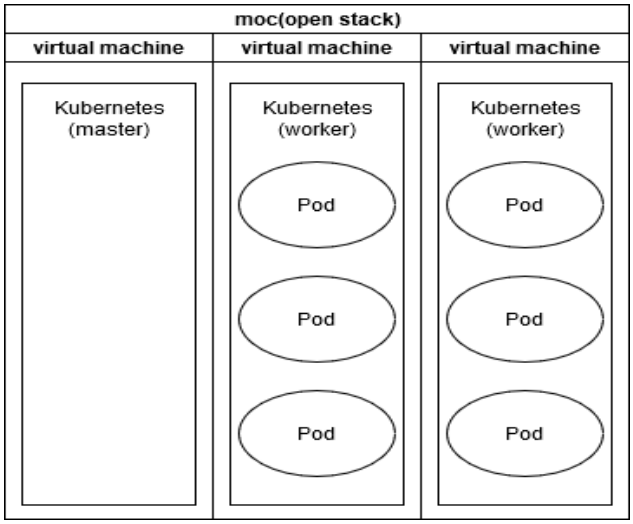
Create-Kubernetes-based-demand-research

Overview

Architecture:



We deployed all front end and middleware on openshift and backend on openstack.



This figure showed how sid will work on moc, sid will be deployed as an application in kubernetes' pod.

This documentation includes the tech we used in the project. including:

Backend:

1. Kubernetes
2. Ansible
3. Kubespray

4. OpenStack
5. Terraform

Frontend and middleware:

6. Node WebServer
7. Redis Job Queue
8. MongoDB for Job and User State
9. Node Worker for communicating with the Kubernetes cluster

Backend:

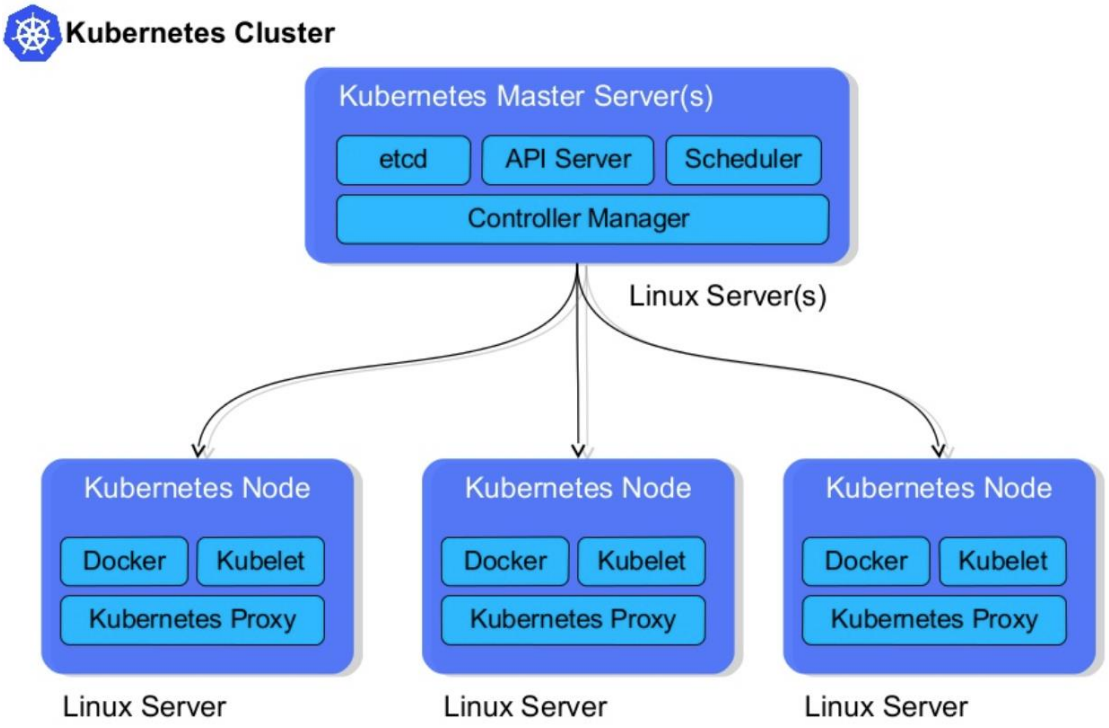
1. Kubernetes

1.1. Overview

We use Kubernetes to manage all the containers on which sid can work.

Kubernetes is an open-source system that allows organizations to deploy and manage containerized applications like platforms as a service (PaaS), batch processing workers, and microservices in the cloud at scale. Through an abstraction layer created on top of a group of hosts, development teams can let Kubernetes manage a host of functions--including load balancing, monitoring and controlling resource consumption by team or application, limiting resource consumption and leveraging additional resources from new hosts added to a cluster, and other workflows.

1.2. Kubernetes Architecture (Master, Worker)



The Kubernetes master is responsible for maintaining the desired state for your cluster. The master can also be replicated for availability and redundancy. When you interact with Kubernetes, eg. via the kubectl command-line interface, you're communicating with the master. The worker nodes in a cluster are the machines (VMs, physical servers, etc) that run your applications and cloud workflows. The master controls each node; you'll rarely interact with nodes directly.

2. Terraform

2.1 Overview

We use terraform to automatically create new virtual machine on openstack.
Terraform is an open-source infrastructure as code software tool created by HashiCorp. It enables users to define and provision a datacenter infrastructure using a high-level configuration language known as Hashicorp Configuration Language (HCL), or optionally JSON.

2.2 Deploying a compute cluster in OpenStack via Terraform

a) Log in to the OpenStack dashboard, create Application Credentials and save clouds.yaml.(we need to do it because terraform only support Single sign-on while we have no id.)
b) Create a providers.tf file with the following contents

```
provider "openstack" {
  cloud = "openstack"
}
```

c) Create a key pair in openstack, save the key.pem file.
you can find the public key by running the command

```
$ ssh-keygen -y -f /path/to/key.pem
```

d) Create a new file called main.tf with the following structure. In public_key, write the complete value of your public key.

```
resource "openstack_compute_keypair_v2" "my-cloud-key" {
  name       = "my-key"
  public_key = "ssh-rsa AAAAB3Nz..."
}
```

e) Configure openstack instance in main.tf in this format

```
resource "openstack_compute_instance_v2" "test" {
  name            = "test-vm"
  image_name      = "denbi-centos7-j10-2e08aa4bfa33-master"
  flavor_name     = "m1.tiny"
  key_pair        = "${openstack_compute_keypair_v2.my-cloud-
key.name}"
  security_groups = ["default"]
```

```
  network {
    name = "public"
  }
}
```

d) Deploy new cluster by running these commands

```
brew install terraform
terraform init
terraform plan
terraform apply
```

3. Ansible

3.1 Overview

We use ansible to automatically deploy Kubernetes on openstack virtual machine.
Ansible is a configuration management platform that automates storage, servers, and networking. When you use Ansible to configure these components, difficult manual tasks become repeatable and less vulnerable to error.

4. Kubespray

4.1 Overview

We use kubespray to automatically deploy Kubernetes on openstack virtual machine.
Kubespray is a composition of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes clusters configuration management tasks.

4.2 Kubernetes on Openstack with Terraform

Just follow kuberspray's github
https://github.com/kubernetes-sigs/kubespray/tree/master/contrib/terraform/openstack

a) Add ssh key

```
$ eval $(ssh-agent -s)
$ ssh-add ~/.ssh/id_rsa
```

b) Open group_vars/all/all.yml and add value for "openstack_password" and set

```
cloud_provider: openstack.
```

c) configure cluster variable

Open sample-inventory/cluster.tfvars, and modify these variables:

```
network_name = "<network>"

external_net = "<UUID>"
```

```
subnet_cidr = "<cidr>"

floatingip_pool = "<pool>"

bastion_allowed_remote_ips = ["0.0.0.0/0"]
```

d) Deploy Kubernetes
cd back to terraform-backend-sid directory, and run "ansible-playbook --become -i inventory/$CLUSTER/hosts cluster.yml"

Frontend and middleware:

Frontend and middleware include:

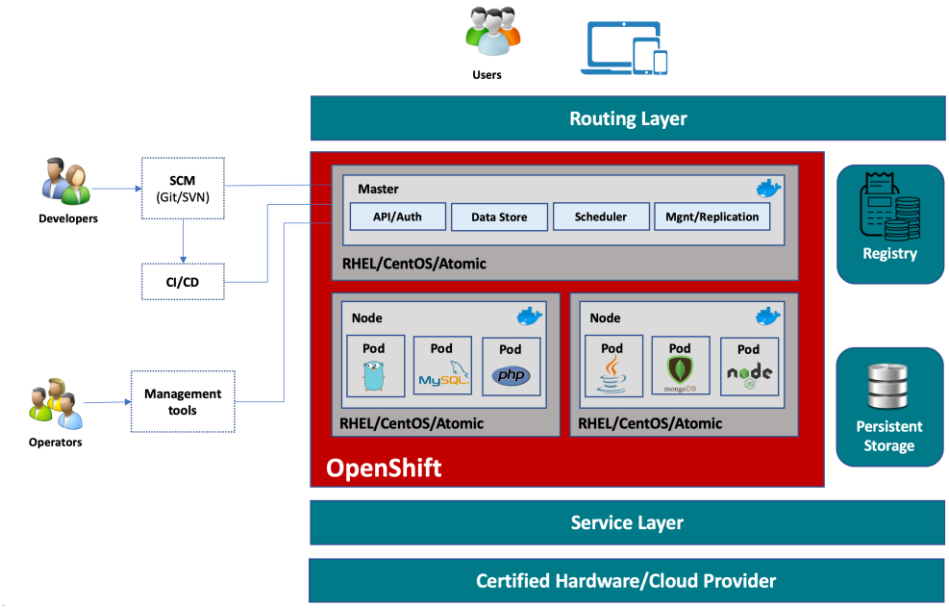
1. Node WebServer
2. Redis Job Queue
3. MongoDB for Job and User State
4. Node Worker for communicating with the Kubernetes cluster

The HMDC team was previously deploying this with Heroku (another cloud provider). Our implementation focuses on getting this to work on the Mass Open Clouds Openshift cluster. MongoDB and Redis are commonly used containers that are publicly available on the Docker Registry. Docker was used to build, tag, and push the two Node server container images to the Openshift Docker Registry.
After being containerized, these 4 components are bundled into a single pod to allow a shared network between the components. This deployment structure is done with the Terraform Kubernetes provider which allows you to define the deployment as a configuration style file. Terraform allows us to quickly bring the whole application up/down.

Openshift

Overview

We deployed the sid front end on Openshift.
OpenShift is a platform as a service (PaaS) that leverages the core concepts of Kubernetes and packages them in a neat way for developers to deploy applications on the cloud. In short, it's a modded Kubernetes, and accepts kubctl commands.



Redis

Overview

We take Redis as our task queue.
Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker.

Job queue

When handling requests from web clients, sometimes operations take more time to execute than we want to spend immediately. We can defer those operations by putting information about our task to be performed inside a queue, which we process later.

MongoDB

Overview

We use MongoDB to store job and user state.
MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL).

Technical

Most of the deployment is done with Terraform; however, you must first push build and push the Node servers to the MOC's Docker Registry.

Prerequisites

- In order to push to the Docker registry on MOC you need to request access via a ticket
- MOC provides X509 self signing certs so the following must be added to **/etc/docker/daemon.json:**

{
"insecure-registries" : ["docker-registry-default.k-apps.osh.massopen.cloud"]
}

- The Kubernetes provider in Terraform needs a login token generated from the openshift frontend (see image below) or the Openshift CLI tool using (oc whoami -t). This token should be placed in **$PROJECT_DIR/terraform_oc/provider.tf**

Project Setup

```
    #install the two git projects
git clone $PROJECT_GIT_URL
git clone $SID_PROJECT_URL
#move the docker files to sid
cp $PROJECT_DIR/scripts/docker/* $SID_PROJ_DIR/

#cd to the sid directory
cd $SID_PROJ_DIR

#build, and tag images to MOC
docker build -t <MOC_DOCKER_URL>/<PROJECT_NAME>/gulp:latest -f
Dockerfile-gulp .
docker build -t <MOC_DOCKER_URL>/<PROJECT_NAME>/worker:latest -f
Dockerfile-worker .

#push images to MOC
docker push <MOC_DOCKER_URL>/<PROJECT_NAME>/gulp:latest
docker push <MOC_DOCKER_URL>/<PROJECT_NAME>/worker:latest

#cd back to project
cd $PROJECT_DIR/terraform-oc/
```

```
#modify provider token
Vim provider.tf
```

Using Terraform

After running the terraform apply command in the following folder the openshift console will display your deployment.

```
    cd $PROJECT_DIR/terraform-oc
#run terraform init to download provider files
terraform init
#terraform apply will apply the current config
terraform apply
#terraform destroy will remove the current config
terraform destroy
```

To-Do

In order to complete this deployment we would have to combine our terraform files into one config. This will allow for the Openstack provider to pass the CA cert and token to the Kubernetes provider. This cert and token are required by the worker to communicate with the backend. I already started doing this by introducing two terraform modules. Modules define resources as groups but still allow for communication. Additionally, we must finish setting up the env variables of the containers to have them properly run.