**Provisioning AWS Infrastructure**

Please follow this doc if bastion server was already provisioned as described in “Bastion host initial setup” of the [Guide](https://github.com/360youlun/devops/blob/master/bastion/README.md)

**Installing salt master**

Login to bastion server with devops role account:

$ ssh <devopsusername>@<bastion\_ip\_or\_dnsname> -p 2222

Install salt-master:

$ sudo add-apt-repository ppa:saltstack/salt

$ sudo apt-get install python-software-properties

$ sudo apt-get install software-properties-common

$ sudo apt-get update

$ sudo apt-get install salt-master salt-cloud salt-ssh salt-api

$ sudo su -

# cd devops

# cp -p /etc/salt/master /etc/salt/master.bak

# cp salt/salt-master/master /etc/salt/master

# service salt-master restart

***No more required, added a script for installing salt-master to “Bastion host initial Setup” Guide***

**Configuring salt-cloud**

How-to on Configuring salt-cloud and provisioning servers(*Let's skip using salt-cloud for now coz it does not work in China region currently. Will be fixed in next releases probably.* )

# cp -p /etc/salt/cloud /etc/salt/cloud.bak

# cp -p salt/salt-master/cloud /etc/salt/cloud

# cp -p salt/salt-master/cloud.profiles /etc/salt/cloud.profiles

# chmod 640 /etc/salt/cloud

Edit /etc/salt/cloud and update next parameters:

<SALT MASTER PUBLIC IP>

<AWS ACCESS KEY ID>

<AWS SECRET ACCESS KEY>

# mkdir /opt/sshkeys

# chmod 770 /opt/sshkeys

# cp salt/salt-master/pillar/private/keys/ylly-devops.pem /opt/sshkeys/

# chmod 600 /opt/sshkeys/ylly-devops.pem

Run salt-cloud to privision any required server.

For example, to provision RabbitMQ server, run command:

# salt-cloud -p base\_rmq\_medium rmq

If salt cloud configuration is correct, the command above will create new AWS instance 'rmq', install salt-minion and automatically authenticate it to salt-master and then start provisioning rmq server if that found in top.sls config

**How-to on Installing salt-minion on any server manually**

Login to aws server prepared for running some of role(rmq, web1, worker1, etc.)

$ sudo add-apt-repository ppa:saltstack/salt

$ sudo apt-get install python-software-properties

$ sudo apt-get install software-properties-common

$ sudo apt-get update

$ sudo apt-get install salt-minion

Edit salt minion config /etc/salt/minion and change 2 parameters:

master: <MASTER>

id: <ID>

Change <MASTER> to real(elastic) IP of salt master, or to it's public DNS name, e.g. bastion.ylly.com

Change <ID> to current minion id, e.g. web1

Save changes and close config /etc/salt/minion

Restart service salt-minion.

Open console on bastion host and run `$ sudo salt-key -L` to check if id of minion apperared in Unaccepted Keys:

If there is id of target minion, accept it's key by next command:

$ sudo salt-key -A

After that you can continue with provisioning server by running highstate:

$ sudo salt-call state.highstate

**Creating Elasticache Cluster with 1-node Redis backend**

Login to AWS console as devopsonduty user.

Go to Services -> ElastiCache -> Launch Cache Cluster

Select Engine -> Redis -> Next

Engine Version: 2.8.6

Port: 6379

Enable Replication: unchecked

Cluster Name: redis-prod

Node Type: cache.m3.medium

Press Next

On “Configure Advanced Settings” screen:

Availability Zone: cn-north-1a

VPC Security Groups: sg-redisfromvpc

Enable Automatic Backups: checked

Backup Retention Period: 7 days

Backup Window: *to be discussed*

Maintenance Window: *to be discussed*

Press Next -> Launch Cache Cluster

**Creating RDS instances and importing production data**

Login to AWS console as devopsonduty user.

Go to Services -> RDS -> Launch a DB instance -> select MySQL.

In the “DB Engine Version” select 5.6.21

In the “DB Instance Class” select db.t2.medium

In the “Multi-AZ Deployment” select No

In “Storage Type” select Provisioned IOPS(SSD)

In “Allocated storage” enter: 100Gb

In “Provisioned IOPS” enter: 1000

In “DB Instance Identifier” enter: cruise

In “Master Username” enter: cruise

In “Master Password” and “Confirm Password” enter new secure password and save it somewhere(Passpack)

Press Next Step.

On Configure Advanced Setting Screen use next parameters:

VPC: Default VPC

Subnet Group: Default

Publicly Accessible: Yes

Availability Zone: cn-north-1a

VPC Security Groups: sg-mysqlfromall

Database Name: cruise

Backup Retention Period: 7 days

Backup Window: Let’s discuss, I think should night time of some day when less activity/users’ traffic. Anyway this can be changed later.

Auto minor version upgrade: Yes

Maintanance Window: Should be like Backup Window +1 hour

Wait for instance status to become “available”.

After RDS instance provisioned, login to it using mysql client and create a new database cruise:

$ mysql -u cruise --password=<mysql\_pwd> -h <rds\_endpoint>

where <mysql\_pwd> password used during RDS instance creation and <rds\_endpoint> can be taken from AWS console

mysql> create database cruise;

Query OK, 1 row affected (0.34 sec)

mysql> quit

Bye

Prepare dump with production data and save it locally as cruise.sql

Import data from dump to RDS instance:

$ mysql -u cruise --password=<mysql\_pwd> -h <rds\_endpoint> cruise < cruise.sql

Create Read Replica for RDS instance cruise.

Go to AWS console -> Services -> RDS. Right click on cruise RDS instance and select “Create Read Replica”.

In the “DB Instance Class” select db.t2.small

In “Allocated storage” enter: 100Gb

In “Provisioned IOPS” enter: 1000

In “Read Replica Source” select **cruise**

In “DB Instance Identifier” enter: **cruise-rr**

Availability Zone: cn-north-1a

Press “Create Read Replica” and for status of instance **cruise-rr** became “available”

Go back to bastion host and install rdscli:

$ ssh <devopsusername>@<bastion\_ip\_or\_dnsname> -p 2222

$ sudo su -

# wget <http://s3.amazonaws.com/rds-downloads/RDSCli.zip>

# unzip RDSCLI.zip

Make a note of unpacked directory with RDSCLI (RDSCli-1.19.002 currently)

# touch .aws/credential-file

Edit .aws/credential-file and add 2 lines to it with appropriate valuse of access key id and secret access key id:

AWSAccessKeyId=<ACCESS KEY ID>

AWSSecretKey=<SECRET ACCESS KEY>

Add next 5 lines to end of /root/.bashrc file and save it:

export AWS\_RDS\_HOME=$HOME/RDSCli-1.19.002

export AWS\_CREDENTIAL\_FILE=$HOME/.aws/credential-file

export PATH=$PATH:$AWS\_RDS\_HOME/bin

export EC2\_REGION=cn-north-1

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64/jre

# source ~/.bashrc

Check if rds cli was configured correctly by next command:

# rds-describe-db-instances

You should see output for 2 existing instances.

Run script to apply utf8 settings on RDS instances:

# . ~/devops/aws/rds/rds\_utf8\_setup.sh

The script will update both RDS instances settings to utf8 and reboot them to take effect.

***Note***: don’t use rdscli from Ubuntu’s repository as it outdated version and may not function properly.

**AWS Security Groups**

|  |  |  |  |
| --- | --- | --- | --- |
| Group Name | Rules | Description | Servers affected |
| sg-nginxfromall | ports 80  source 0.0.0.0/0 | Allows http connections on port 80 | worker1  web1 |
| sg-rmqfromvpc | ports 5672,15672  source 172.31.0.0/16 | Allows connections to RabbitMQ server from VPC CIDR addresses | rmq |
| sg-saltfromall | ports 4505,4506  source 0.0.0.0/0 | Allows connections to salt-master ports from any source | bastion |
| sg-sshfromall | ports 2222  source 0.0.0.0/0 | Allows ssh connections to bastion on port 2222 from any source | bastion |
| sg-uwsgifromall | ports 8000  source 0.0.0.0/0 | Allows http connections to uWSGI port from any source | web1 |
| sg-redisfromvpc | ports 6379  source 172.31.0.0/16 | Allows connections to Elasicache Redis cluster from VPC CIDR | redis-prod |
| sg-sshfrombastion | ports 22  source 172.31.0.10/32 | Allows ssh connections to any aws server from bastion only | All AWS servers(EC2) except bastion |
| sg-mysqlfromall | ports 3306  source 0.0.0.0/0 | Allows connections to production RDS MySQL instances | RDS instances:   * cruise * cruise-rr |
| sg-flowerfromall | ports 8080  source 0.0.0.0/0 | Allows http connections to flower | worker1 |

**AWS servers preparation order**

1. Setup and configure bastion host and security groups as per [Guide](https://github.com/360youlun/devops/blob/master/bastion/README.md)
2. Create new Elasticache Redis instance(cache.m3.medium) as described above. After Redis cluster is created, take Endpoint of cluster and update it’s value as **redis\_host** in <https://github.com/360youlun/pillar/blob/master/pillar/private/prod-data.sls>.
3. Create RDS instances and import data from production dump. Take mysql instances endpoints and update 2 **host** values in databases/replicas sections of https://github.com/360youlun/pillar/blob/master/pillar/private/prod-data.sls
4. Create new EC2 instance with name rmq(t2.medium) and configure salt-minion with name rmq for it. Associate security groups “sg-sshfrombastion”, “sg-rmqfromvpc” to this instance during creation.

Perform highstate on it and make sure it finished without errors.

1. Write down Private DNS name of rmq host and update it in <https://github.com/360youlun/pillar/blob/master/pillar/private/rmq-data.sls> as BROKER\_URL
2. Create new EC2 instance with name worker1(c3.large) and configure salt-minion with name worker1 for it.

Associate security groups “sg-sshfrombastion”, “sg-nginxfromall”, “sg-flowerfromall” to this instance during creation.

Perform highstate on it and make sure it finished without errors.

1. Associate new Elastic IP to previously created worker1 server and create(update) DNS records for it (worker1.c1.ylly.io, flower.ylly.io). Update this new DNS record(instead of worker1.c1.360yln.com) in <https://github.com/360youlun/pillar/blob/master/pillar/private/workerserver-data.sls>
2. Create new EC2 instance with name web1(m3.medium) and configure salt-minion with name web1 for it.

Associate security groups “sg-sshfrombastion”, “sg-nginxfromall”, “sg-uwsgifromall”, to this instance during creation.

1. Associate new Elastic IP to previously created web1 server and create(update) DNS records for it (production.ylly.com, en.ylly.com).

**Discussion Topics**

1. Zabbix Server.

There are 2 possible ways to move zabbix from old aws account:

* Create dump of MySQL database for current Zabbix. Install and configure new Zabbix server. Import data from dump. Re-configure Zabbix to handle new AWS servers.
* Move the whole Zabbix server AMI and re-configure for new address. Re-configure Zabbix to handle new AWS servers.

1. Newrelic instance(with agents for monitoring AWS servers). Currently cannot be installed from original Newrelic AMI because it’s missing in China region currently. The only possible way is to install clean Ubuntu and configure all the agents/plugins manually by manuals(or by writing salt configuration for it). The only exception for aws monitoring plugin is that can be run only when old one(current) is stopped(not possible to monitor 2 different AWS accounts from one Newrelic account). Also no guarantee that all of Newrelic plugins will work in China region(strong intuition based on experience😃 ).
2. Security Groups

sg-saltfromall - for better security we should avoid allowing connections to salt-master from worldwide. If all our external(not AWS) servers(like staging, varnish, etc.) have static IPs, then we should restrict rules by those IPs only + VPC CIDR.

sg-uwsgifromall - not sure if someone requies direct access to upstream uwsgi server. If not, then we don’t need this group

sg-mysqlfromall - We have access to MySQL ports on production allowed from worldwide, and that is not secure too. Because dumps are prepared from worker server, do we need to access production MySQL directly for some reason? If not, then let’s restrict access to MySQL on port 3306 from VPC CIDR only. If we will require to access MySQL directly sometime for some reason, then we can temporarily add a rule for that.

1. ...