horizontal line

Versa Auto Deployment

|  |  |  |
| --- | --- | --- |
| **Version #** | **Date** | **By** |
| 1.0 | 10/09/2018 | Sampathkumar and Sathish |
| 1.1 | 1/10/2018 | PP |
|  |  |  |

# Table of Contents

[**Table of Contents**](#_7noxn4wpv78a) **2**

[Introduction:](#_9vtha623wb8m) 3

[Auto Deployment on Versa QCOW](#_8l1thl2ozb6a) 3

[Prerequisites:](#_3ccuhx7q2q3z) 3

[Single Node Deployment: Need 1 VM’s with Versa QCOW Installed](#_8hbubr7g2eks) 3

[Multi Node Deployment: Need 3 VM’s with Versa QCOW Installed](#_2m715rt9fkwh) 3

[To download versa QCOW2 for installing saas & shop;](#_xsqmah79qqso) 3

[To download versa QCOW2 for installing global haproxy(only for multi node) & GLUU authentication;](#_gy8hh7qqsrc5) 3

[Auto Deploy Script using Ubuntu 16.04](#_8rrr19xjhaqa) 4

[Install Docker CE](#_kbaxnjlv9kd9) 4

[Install Python Packages](#_xivy5fm92n70) 4

[PIP Packages Used](#_8t7fwurp8rl4) 5

[Auto Deployment Flow Diagram](#_9rqs8bbr183j) 6

[**Deploy**](#_bi37a24vajpg) **7**

[Check Prerequisite](#_3ld3am2waf40) 7

[Multi Node Deployment](#_h77eh5g7mb50) 7

[Single Node Deployment](#_s3fnbzb2b9tf) 9

[Setup Auth Server if required](#_qmts84wqsz35) 11

[**Validate Deployment**](#_8yspok18x7pg) **12**

[Kafka-Zookeeper Check](#_th62fd3s9wzv) 12

[Check Zookeeper and Broker Connectivity](#_c3soyh5sfpqh) 13

[MongoDB Check](#_81ie9lmuxx87) 16

[Dashboard Check](#_q5d3kk8he2u3) 19

[**Appendix**](#_bws8x364vlve) **23**

[Auto Deployment Script](#_vvo3ugrdirav) 23

[Main Method](#_9qhb7t2dpc7) 23

[Single/Multi Node Deployment](#_dbrsan4uqkez) 23

[Auto Deployment Flow](#_58j2nrd121ln) 24

[Modifying configuration Values](#_nsh61l49xhb4) 24

[Initiating the Deployment](#_z0nuo99r66mw) 25

[Initiate Swarm Cluster](#_zaeztdh8nmg5) 25

[Configure Mongo](#_oocd09jle0jh) 26

[Configure Kafka](#_n6yqdpv4r0re) 29

[Configure Mysql](#_ivd0ehema4si) 30

[Configure Rsync](#_iamw8uvg7u71) 31

[Deployment](#_btp9ucwdhc4) 33

[Configure Logstash](#_wlyltasbf9c9) 33

[Configure HaProxy](#_if5bcr557knj) 34

## Introduction:

Deployment can be done using versa builded QCOW2 and customer OS. with respect to customer os, please follow auto deployment using ubuntu16.04 section.

## Auto Deployment on Versa QCOW

### Prerequisites:

### Single Node Deployment: Need 1 VM’s with Versa QCOW Installed

### Multi Node Deployment: Need 3 VM’s with Versa QCOW Installed

#### **To download versa QCOW2 for installing saas & shop;**

**From: Outside Versa:** <https://saasrepo.versa-networks.com/repository/packages/ubuntu1604-versa-saas/ubuntu1604-docker-versa.qcow2.tar>

**Using Versa VPN/Intra Network:**

<https://10.48.27.10/repository/packages/ubuntu1604-versa-saas/ubuntu1604-docker-versa.qcow2.tar>

Username: download

Password: download@123

#### **To download versa QCOW2 for installing global haproxy(only for multi node) & GLUU authentication;**

**From: Outside Versa:** <https://saasrepo.versa-networks.com/repository/packages/ubuntu1604-versa-webportal/ubuntu1604-versa-webportal.qcow2.tar>

Username: download

Password: download@123

**Using Versa VPN/Intra Network:**

<https://10.48.27.10/repository/packages/ubuntu1604-versa-webportal/ubuntu1604-versa-webportal.qcow2.tar>

## 

## Auto Deploy Script using Ubuntu 16.04

### 

### Initial Requirement

Create an user with the *username*: **versa** and *password*: **versa123**

If it is multi node, we need to create the same username and password for all the VM and need to follow the below steps in **all three nodes**.

### 

### Directory Structure

The directory structure should be the following:

Home Directory : Should be **/home/versa**

Deployment Stack : You can download the [deployment stack from here](https://drive.google.com/open?id=1QRkkyLRv_ihG-fJlu3XF0yrZrIL-f4jf) and place it in **deployment\_stack**.

***Note: The downloaded directory should be renamed to “deployment\_stack”***

Docker Images: You can download the [versa docker images from here](#_n4stn3nhq9e6) and place it inside **/home/versa/docker\_images** directory.

|  |
| --- |
| **/home/versa /home/versa/deployment\_stack /home/versa/docker\_images /home/versa/versa.conf /home/versa/versalog** |

**Create “versa.conf” and “versalog” file in /home/versa directory and add the below content.**

##### **#versa.conf**

|  |
| --- |
| # Add these lines module(load="imudp") **input**(**type**="imudp" port="514" ruleset="application") **input**(**type**="imudp" port="515" ruleset="mongo-kafka")  ruleset(name="application") {  # Versa API - Saas Backend logs  local0.\* /**var**/**log**/versa\_api.log   # Versa Dashboard - Saas Frontend logs  local1.\* /**var**/**log**/versa\_dashboard.log   # Versa Shop - Ecommerce logs  local2.\* /**var**/**log**/versa\_shop.log   # Versa Proxy - HAProxy logs  local3.\* /**var**/**log**/versa\_haproxy.log   # Versa - Node Socket logs  local4.\* /**var**/**log**/versa\_nodesocket.log   # Versa - Stream Processor logs  local5.\* /**var**/**log**/versa\_streamprocessor.log   # Versa - Device Facing API logs  local6.\* /**var**/**log**/versa\_device\_facing\_api.log }  ruleset(name="mongo-kafka") {  # Versa - Kafka Zookeeper logs  local0.\* /**var**/**log**/versa\_kafka.log   # Versa - mongodb logs  local1.\* /**var**/**log**/versa\_mongo.log   # Versa - logstash logs  local2.\* /**var**/**log**/versa\_logstash.log   # Versa - Mysql logs  local3.\* /**var**/**log**/versa\_mysql.log } |

##### #versalog

|  |
| --- |
| /**var**/**log**/versa\_api.log /**var**/**log**/versa\_dashboard.log /**var**/**log**/versa\_shop.log /**var**/**log**/versa\_haproxy.log /**var**/**log**/versa\_nodesocket.log /**var**/**log**/versa\_streamprocessor.log /**var**/**log**/versa\_device\_facing\_api.log /**var**/**log**/versa\_kafka.log /**var**/**log**/versa\_mongo.log /**var**/**log**/versa\_logstash.log /**var**/**log**/versa\_mysql.log {  missingok  sharedscripts  daily  **rotate** 5  size 5M  copytruncate  **compress**  delaycompress  create 0600 root root } |

### Install Docker CE

Run the below ***docker.sh*** script to install Docker CE in Ubuntu 16.04.

|  |
| --- |
| **#!/bin/bash**  *# Update* *#sudo apt-get update*  echo "Install packages to allow apt to use a repository over HTTPS" sudo apt-get install \  apt-transport-https \  ca-certificates \  curl \  software-properties-common -y  echo "Add Dockerâ€TMs official GPG key" sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  echo "Add fingerprint" sudo apt-key fingerprint 0EBFCD88  echo "Set up the stable repository" sudo add-apt-repository \  "deb [arch=amd64] https://download.docker.com/linux/ubuntu \  $(lsb\_release -cs) \  stable"  *#Update again* sudo apt-get update  echo "Install latest docker" sudo apt-get install docker-ce -y  sudo apt-get install python-pip |

### Install Python Packages

We need some pip packages to run the auto deployment script. All the required pip packages has been listed in the requirements.txt file which located in /home/versa/deployment\_stack/auto\_deploy

Run the below command to install all the pip packages,

|  |
| --- |
| $ cd /home/versa/deployment\_stack/auto\_deploy ~/deployment\_stack/auto\_deploy$ pip install -r requirements.txt |

#### ***PIP Packages Used***

*To run the auto deployment script we need to have the below list of python packages in the primary VM (Primary VM: The VM where we are going to run the deployment script)*

1. ***jinja2==2.10*** *- Template Manager to modify env variables. This will used to modify the configuration values of Dashboard, API, Stream Processor, Node Socket.*
2. ***paramiko==2.4.1*** *- SSH Client to perform SSH process. This will be used to login to other manager IP’s through SSH and execute shell command.*
3. ***pyyaml==3.13*** *- YAML Package. In this Auto Deploy Script all the user inputs will be stored in yml file as a configuration file. With the help of this package we can get values from yml file as key value array pairs*
4. ***subprocess32****==3.5.2 - To execute shell command within Python Script. This would execute shell script to the current manager (where the python script is running).*
5. ***scp****==0.11.0 - This module will helps to copy file from one swarm node to another swarm node.*

### To download versa latest docker images

#### **From: Outside Versa:**

<https://saasrepo.versa-networks.com/repository/packages/versa-saas-docker-image/versa_1.0R/versa_1.0R.tar.gz>

Username: download

Password: download@123

#### **Using Versa VPN/Intra Network:**

<https://10.48.27.10/repository/packages/versa-saas-docker-image/versa_1.0R/versa_1.0R.tar.gz>

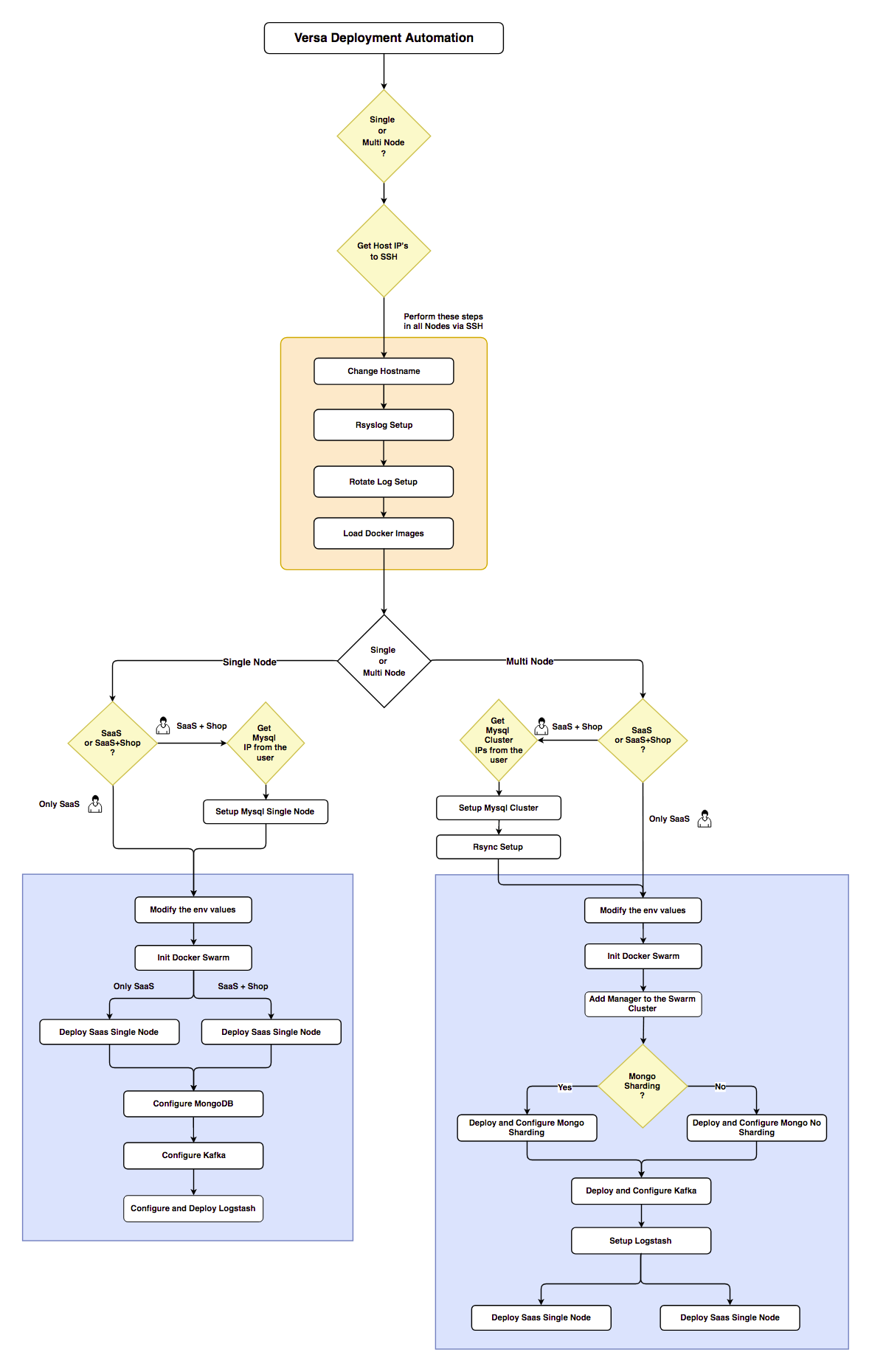
Need to extract the versa\_1.0R.tar.gz **/home/versa/** directory. The versa\_1.0R.tar.gz file should be in docker\_images directory.

The file directory will look like this.

|  |
| --- |
| /home/versa/docker\_images/versa\_1.0R.tar.gz |

## 

## Auto Deployment Flow Diagram



# Deploy

## Check Prerequisite

We need to check all the machines are connected or not. This should be checked in the primary node where the deployment script is running.

|  |
| --- |
| **$** ping <manager\_2\_node\_ip> **$** ping <manager\_3\_node\_ip> **$** ping <webportal\_node\_ip> |

And before running the auto deployment script, please check the prerequisites are installed in the node.

***Note: This script will only validate the current running machine. It won’t check the prerequisite for another nodes. We need to check this on every machine.***

Run the below command in terminal to check the prerequisites.

|  |
| --- |
| $ cd /home/versa/deployment\_stack/auto\_deploy ~/deployment\_stack/auto\_deploy$ python prerequisite.py  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Success!!! Now you can run setup.py  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |

If the script gives the success message, then you are free to run the auto deploy script.

Follow the below steps to run the auto deploy script. You can find the Inputs & Descriptions [**here**](#_dbrsan4uqkez).

*Note: Currently the deployment script will run only once. We cannot re-run the script.*

## Multi Node Deployment

|  |
| --- |
| $ cd /home/versa/deployment\_stack/auto\_deploy ~/deployment\_stack/auto\_deploy$ python setup.py  1: SaaS Only  2: SaaS + Shop  Please **enter** deployment type: [1/2] :2 True   1: Multi Node  2: Single Node  Multi node **or** Single node: [1/2] :1 True   Manager IP addresses separated by comma(,) without spaces  The sequence should be manager0, manager1, manager2 **Enter** valid IP Address: 192.168**.1.10**,192.168**.1.11**,192.168**.1.12** ['192.168.1.10', '192.168.1.11', '192.168.1.12']   **Enter** Haproxy Address/App Endpoint: 192.168**.1.13** 192.168**.1.13**   1: No Sharding  2: Sharding  **Enter** mongo cluster type: [1/2] :1 NO\_SHARDING   **Enter** Oauth endpoint without 'http:// (or) https://': oauth.com  oauth.com   **Enter** **Default** Director IP: 205.251**.66.57** **Enter** VD Username: Administrator **Enter** VD Password: \*\*\*\*\*\*\*\* **Enter** Controller 1 IP: 205.251**.66.60** **Enter** Controller 1 Site ID: 1 **Enter** Controller 1 Name: controller-1 **Enter** Controller 2 IP: 205.251**.66.61** **Enter** Controller 2 Site ID: 2 **Enter** Controller 2 Name: controller-2 **Enter** **Default** Provider Name: Provider **Enter** **Default** Provider Site ID: 1 **Enter** Analytics IP: 192.168**.110.110** **Enter** Analytics Port: 1234 **Enter** Speed **Test** IP: 207.47**.61.67** **Enter** AMQP IP: 10.75**.26.150** **Enter** AMQP Port: 5672  ***Verify the below informations*** *Deployment with [SaaS + Shop] Deployment type is : [Multi Node] Manager0 IP Address : [192.168****.1.10****] Manager1 IP Address : [192.168****.1.11****] Manager2 IP Address : [192.168****.1.12****] Haproxy IP : [192.168****.1.13****] App Endpoint : [192.168****.1.13****] Mongo Cluster type is [No Sharding] Oauth endpoint : [<oauth\_endpoint>] Director IP is : [205.251****.66.57****] VD Username is : [Administrator] VD Password is : [\*\*\*\*\*\*\*\*] Controller 1 Name is : [controller-1] Controller 1 IP is : [205.251****.66.60****] Controller 1 Site ID is : [1] Controller 2 Name is : [controller-1] Controller 2 IP is : [205.251****.66.61****] Controller 2 Site ID is : [2]* ***Default*** *Provider name is : [Provider]* ***Default*** *Provider site ID is : [1] Analytics IP is : [192.168****.110.110****] Analytics Port is : [1234] Speed* ***Test*** *IP is : [207.47****.61.67****] AMQP IP is : [10.75****.26.150****] AMQP Port is : [5672]     Are you sure, you want to go with proceed the Deployment? [y/N] y* |

## Single Node Deployment

|  |
| --- |
| $ cd /home/versa/deployment\_stack/auto\_deploy ~/deployment\_stack/auto\_deploy$ python setup.py  1: SaaS Only  2: SaaS + Shop  Please **enter** deployment type: [1/2] :2 True   1: Multi Node  2: Single Node  Multi node **or** Single node: [1/2] :2 False   Manager IP address **Enter** valid IP Address: 192.168**.1.10** 192.168**.1.10**   **Enter** Oauth endpoint without 'http:// (or) https://': oauth.com oauth.com   **Enter** **Default** Director IP: 205.251**.66.57** **Enter** VD Username: Administrator **Enter** VD Password: \*\*\*\*\*\*\* **Enter** Controller 1 IP: 205.251**.66.60** **Enter** Controller 1 Site ID: 1 **Enter** Controller 1 Name: controller-1 **Enter** Controller 2 IP: 205.251**.66.61** **Enter** Controller 2 Site ID: 2 **Enter** Controller 2 Name: controller-2 **Enter** **Default** Provider Name: Provider **Enter** **Default** Provider Site ID: 1 **Enter** Analytics IP: 192.168**.110.110** **Enter** Analytics Port: 1234 **Enter** Speed **Test** IP: 207.47**.61.67** **Enter** AMQP IP: 10.75**.26.150** **Enter** AMQP Port: 5672  ***Verify the below informations...*** *Deployment with [SaaS + Shop] Deployment type is : [Single Node] Manager IP Address : [192.168****.1.10****] App Endpoint : [192.168****.1.10****] Oauth endpoint : [oauth.com] Director IP is : [205.251****.66.57****] VD Username is : [Administrator] VD Password is : [\*\*\*\*\*\*\*] Controller 1 Name is : [controller-1] Controller 1 IP is : [205.251****.66.60****] Controller 1 Site ID is : [1] Controller 2 Name is : [controller-2] Controller 2 IP is : [205.251****.66.61****] Controller 2 Site ID is : [2]* ***Default*** *Provider name is : [Provider]* ***Default*** *Provider site ID is : [1] Analytics IP is : [192.168****.110.110****] Analytics Port is : [1234] Speed* ***Test*** *IP is : [207.47****.61.67****] AMQP IP is : [10.75****.26.150****] AMQP Port is : [5672]*     Are you sure, you want to go with proceed the Deployment? [y/N] y |

## Setup Auth Server if required

Note: Do the the setup if you require OAuth Server for Authentication.

Please refer the document for GLUU OAuth Setup. This IP should be provided while you run setup.py, **Enter Oauth endpoint without 'http:// (or) https://':**

<https://docs.google.com/document/d/16tE8I5P0G6MTas9HXDVQSRwdLwm_1TCFfjyn6Km34W8>

# 

# Validate Deployment

## Kafka-Zookeeper Check

***How to check whether all the Kafka brokers are running properly?***

We can use zookeeper-shell.sh to check the kafka node status. Find container (kafka\_broker) and Login to the container,

|  |
| --- |
| # Get list of Service Name starting with “kafka”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=kafka"  **NAME REPLICAS**  kafka\_broker1 1/1  kafka\_broker2 1/1  kafka\_broker3 1/1  **kafka\_zookeeper1** 1/1  kafka\_zookeeper2 1/1  kafka\_zookeeper3 1/1  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=<Service\_Name>"  **CONTAINER ID NAMES**  81660a404ac1 kafka\_broker1.1.vmjii3ihid82deswtz4jzqz53  **$** sudo docker exec -it <kafka\_broker\_container\_id> bash  $ cd /opt/kafka  $ zookeeper-shell.sh **kafka\_zookeeper1**:2181 <<< "ls /brokers/ids"  Connecting to kafka\_zookeeper1:2181  Welcome to ZooKeeper!  JLine support is disabled  WATCHER::  WatchedEvent state:SyncConnected type:None path:null  **[1, 2, 3]** |

The IDs returned by the command "ls /brokers/ids" are the IDs representing the Kafka brokers that are registered with Zookeeper. If any broker is down, it won’t display the ID of that broker. Check the kafka logs (/var/log/versa\_kafka.log) on the host, of the broker which is down.

## Check Zookeeper and Broker Connectivity

The Kafka service is used by Zookeeper, Logstash, Stream Processor and SaaS API. Find the service names and port given below,

Broker service names: ***[broker1, broker2, broker3]***

Broker port: ***9092***

Zookeeper service names: ***[zookeeper1, zookeeper2, zookeeper3]***

Zookeeper port: ***2181***

***Check connection from Kafka to Zookeeper***, we can use telnet command to check whether the connection with the specific port is working or not. Login to Broker container and verify the connectivity.

|  |
| --- |
| # Get list of Service Name starting with “kafka\_zookeeper”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=kafka\_zookeeper"  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=<Service\_Name>"  **$** sudo docker **exec** -it <kafka\_zookeeper\_id> bash  **$** telnet zookeeper1 2181trying 10.0.\*.11..  Connected to zookeeper1.  Escape character is '^]'. |

If the response comes, then the connection is working fine. In case if the response is not coming or taking more time as expected, run the below command instead.

|  |
| --- |
| $ echo exit | telnet zookeeper1 9092 $ echo $?  0 |

If the response is 0 then the connection is working. If we get 1, then the connection is not working.

Likewise we need to check all the service.

|  |
| --- |
| $ telnet zookeeper2 2181   $ telnet zookeeper3 2181  $ telnet broker1 9092  $ telnet broker2 9092   $ telnet broker2 9092 |

***Check connection from Logstash amqp to Kafka***, login into logstash container.

|  |
| --- |
| # Get list of Service Name starting with “logstash”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=logstash"  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=logstash\_amqp"  $ sudo docker ps | **grep** logstash\_amqp   $ sudo docker **exec** -it <logstash\_container\_id> bash $ telnet broker1 9092  trying 10.0.\*.11..  Connected to broker1.  Escape character is '^]'. |

If the response comes, then the connection is working fine. In case if the response is not coming or taking more time as expected, run the below command instead.

|  |
| --- |
| $ echo exit | telnet broker1 9092 $ echo $?  0 |

If the response is 0 then the connection is working. If we get 1, then the connection is not working.

Likewise we need to check all other service.

|  |
| --- |
| $ telnet broker2 9092  $ telnet broker3 9092   $ telnet zookeeper1 2181   $ telnet zookeeper2 2181   $ telnet zookeeper3 2181 |

Likewise we need to check the connection between ***Stream Processor to Kafka***, ***SaaS API to Kafka*** and ***Nodepolling to Kafka***

***Kafka check sent and receive message: Produce and Consume***

|  |
| --- |
| # Get list of Service Name starting with “kafka”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=kafka"  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=<Service\_Name>"  **$** docker exec -it <Container\_ID\_Kafka> bash  **$** cd /opt/kafka  **# Multi Node #** Produce in SSH terminal 1 **$** bin/kafka-console-producer.sh --broker-list broker1:9092,broker2:9092,broker3:9092 --sync --topic versa-analytics  **#** Consume in SSH terminal 2 **$** bin/kafka-console-consumer.sh --bootstrap-server broker1:9092,broker2:9092,broker3:9092 --topic versa-analytics --from-beginning |

## 

***Zookeeper autopurge.snapRetainCount***, When enabled, ZooKeeper auto purge feature retains the autopurge.snapRetainCount most recent snapshots and the corresponding transaction logs in the dataDir and dataLogDir respectively and deletes the rest. Defaults to 3. Minimum value is 3.

Add these environment variables in kafka-zookeeper.yml in zookeeper1, zookeeper2, zookeeper3 services

|  |
| --- |
| ZOO\_AUTOPURGE\_SNAPRETAINCOUNT: 3 ZOO\_AUTOPURGE\_PURGEINTERVAL: 1 |

## MongoDB Check

***Check MongoDB connectivity***: MongoDB is used by SaaS API and Stream Processor respectively. To verify mongodb is running properly, we have to check the mongodb service with the help of telnet.

To verify mongo replica set run the following command.

|  |
| --- |
| # Get list of Service Name starting with “mongodb”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=mongodb"  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=<Service\_Name>"   $ sudo docker **exec** -it <mongodb1\_container\_id> bash $ mongo  mongo> **use** admin; Switched to admin mongo> db.auth("admin", "versa@87"); 1 mongo> rs.status(); {  "set" : "set1",  "date" : ISODate("2018-08-31T16:09:58Z"),  "myState" : 2,  "members" : [  {  "\_id" : 0,  "name" : "mongo1:27017",  "health" : 1,  "state" : 1,  "stateStr" : "PRIMARY",  "uptime" : 770622,  "optime" : {  "t" : 1294848597000,  "i" : 13  },  "optimeDate" : ISODate("2018-08-31T16:09:57Z"),  "lastHeartbeat" : ISODate("2018-08-31T16:09:57Z")  },  {  "\_id" : 1,  "name" : "mongo1:27017",  "health" : 1,  "state" : 2,  "stateStr" : "SECONDARY",  "uptime" : 770624,  "optime" : {  "t" : 1294841397000,  "i" : 56  },  "optimeDate" : ISODate("2018-08-31T14:09:57Z"),  "lastHeartbeat" : ISODate("2018-08-31T16:09:57Z")  },  {  "\_id" : 2,  "name" : "mongo2:27017",  "health" : 1,  "state" : 2,  "stateStr" : "SECONDARY",  "uptime" : 770624,  "optime" : {  "t" : 1294841397000,  "i" : 56  },  "optimeDate" : ISODate("2018-08-31T14:09:57Z"),  "lastHeartbeat" : ISODate("2018-08-31T16:09:57Z")  }  ],  "ok" : 1 } |

Find the mongodb service names and port given below,

Mongo service names: ***[mongo1, mongo2, mongo3]***

Mongo port: ***27017***

|  |
| --- |
| # Get list of Service Name starting with “mongodb”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=mongodb"  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=<Service\_Name>"   $ sudo docker **exec** -it <mongo1\_container\_id> bash $ telnet mongo2 27017  trying 10.0.\*.\*\*..  Connected to mongo2.  Escape character is '^]'. |

If the response comes, then the connection is working fine. In case if the response is not coming or taking more time as expected, run the below command instead.

|  |
| --- |
| $ echo exit | telnet mongo2 9092 $ echo $?  0 |

If the response is 0 then the connection is working. If we get 1, then the connection is not working.

Likewise we need to check all other service.

|  |
| --- |
| $ telnet mongo3 27017 |

Once verifying all the mongo services, we need to verify the connection with SaaS API and Stream Processor aswell. To check the connection with api run the below command.

|  |
| --- |
| # Get list of Service Name starting with “mongodb”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" --filter "name=mongodb"  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=<Service\_Name>"  $ sudo docker **exec** -it <mongo1\_container\_id> bash  $ telnet api 80  trying 10.0.\*.\*\*..  Connected to api.  Escape character is '^]'. |

If the response comes as expected, then the connection between mongo and api is working fine. Likewise we need to test the connection from ***Mongo to Stream Processor***,

## Dashboard Check

**Checking Dashboard Connectivity**: Dashboard is used by SaaS,and Nodesocket and GLUU (Oauth Server).

Find the API details given below.

**versa\_api : GLUU**

**Port : 80**

The dashboard is using versa\_api (GLUU). To check connection of versa\_api from the dashboard, run the below command.

|  |
| --- |
| # Get list of Service Name starting with “versa\_dashboard”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" | grep dashboard  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=versa\_dashboard"   $ sudo docker **exec** -it <dashboard\_container\_id> bash $ telnet versa\_api 80  trying 10.0.\*.11..  Connected to versa\_api.  Escape character is '^]'. |

If the response comes as expected then the connection is working fine.

Likewise we need to check all other service.

|  |
| --- |
| $ telnet versa\_api 80  $ telnet versa\_node\_socket1 80  $ telnet versa\_node\_socket2 80 |

And we need to verify the siteminder portal by simply accessing siteminder URL with the help of curl command. Now we are going to test Siteminder oidc token URL with curl command. The response should be the json like values.

|  |
| --- |
| $ curl https://<Oauth\_End\_Point>/oxauth/restv1/token  {"error\_code":"invalid\_request","error\_description":"GET request is not allowed."} |

If the response is comes as above, then the siteminder connection is working find. If we get any timeout error or something, then the connection is not working between SaaS API and Gluu.

**Device Facing API Check**

***Checking Device Facing API***: Device facing API’s which directly communicates the device (Versa Director, Controller and Analytics).

To verify the connection between SaaS API and Device facing API’s we need to use telnet and ping commands.

When comes to Device facing API there are four API addresses, that we should check the connectivity.

* Director IP
* Controller 1
* Controller 2
* Analytics IP

We need to check the connection with all the api services (versa\_api, versa\_api1, versa\_api2). We can find the IP informations in the ***device\_facing\_api.yml*** (/home/versa/deployment\_stacks/stacks/configs)

We can also verify the device facing API by running curl command inside the API container.

|  |
| --- |
| # Get list of Service Name starting with “versa\_api”  **$** sudo docker service ls --format "table {{.Name}}\t{{.Replicas}}" | grep versa\_api  # Get list of container of a particular service  **$** sudo docker ps --format "table {{.ID}} \t {{.Names}} " --filter "name=versa\_api2"  **$** docker exec -it <Versa\_API\_Container\_ID> bash  **$** curl -X GET http://devicefacingapi/api/v1.0/orgs/org/ |

Running the ping command to verify the IP is open.

|  |
| --- |
| $ sudo docker container ls | **grep** api   $ sudo docker exec -it <api\_container\_id> bash $ ping -c 4 205.251.\*\*.\*\* PING google.com (205.251.\*\*.\*\*) 56(84) bytes of data. 64 bytes **from** 205.251.\*\*.\*\*: icmp\_seq=1 ttl=53 time=44.5 ms 64 bytes **from** 205.251.\*\*.\*\*: icmp\_seq=2 ttl=53 time=77.5 ms 64 bytes **from** 205.251.\*\*.\*\*: icmp\_seq=3 ttl=53 time=55.0 ms 64 bytes **from** 205.251.\*\*.\*\*: icmp\_seq=4 ttl=53 time=183 ms  --- 205.251.\*\*.\*\* ping statistics --- 4 packets transmitted, 4 received, 0% packet loss, time 3004ms rtt min/avg/max/mdev = 44.517/90.168/183.473/55.178 ms |

And need to validate the port by running telnet command

|  |
| --- |
| $ sudo docker container ls | **grep** api   $ sudo docker **exec** -it <api\_container\_id> bash $ telnet 205.251.\*\*.\*\* 9182  trying 205.251.\*\*.\*\*..  Connected to 205.251.\*\*.\*\*.  Escape character is '^]'. |

If the response comes as expected then the connection is working fine. Likewise we need to test all the *Director IP*, *Controller 1 IP*, *Controller 2 IP* and *Analytics IP*.

# Appendix

## Auto Deployment Script

### 

### Main Method

The main method will be initiating the Auto Deploy Script. Inside the main method user inputs will be collected. Two types of deployment is there, the user inputs will vary based on the deployment type.

1. Single Node Deployment
2. Multi Node Deployment

### Single/Multi Node Deployment

The below are the inputs we are getting from the user for Single/Multi Node Deployment.

|  |  |
| --- | --- |
| **Field** | **Description / Values** |
| Deployment type ***SaaS Only*** or ***SaaS + Shop*** | 1 is for SaaS Only  2 is for SaaS + Shop |
| Single Node / Multi Node | 1 is for Multi Node  2 is for Single Node |
| Manager IP addresses (3 IP Address) | Node IP Addresses  Single Node: IP address of the Running Node  Multi Node: All three node IP addresses, with comma seperated |
| Haproxy Address/App Endpoint | Haproxy IP address  No need for Single Node deployment |
| Mongo Cluster Type (***No Sharding*** / ***Sharding***) | 1 is for No Sharding  2 is for Sharding  No need for Single Node deployment |
| Oauth endpoint | Oauth endpoint without http/https |
| Default Director IP | Versa Director IP |
| VD Username | Versa Director Username |
| VD Password | Versa Director Password |
| Controller 1 IP | Versa Controller 1 IP Address |
| Controller 1 Site ID | Versa Controller 1 Site ID |
| Controller 1 Name | Versa Controller 1 Name |
| Controller 2 IP | Versa Controller 2 IP Address |
| Controller 2 Site ID | Versa Controller 2 Site ID |
| Controller 2 Name | Versa Controller 2 Name |
| Default Provider Name | Default Provider Name |
| Default Provider Site ID | Default Provider Site ID |
| Analytics IP | Analytics Server IP Address |
| Analytics Port | Analytics Server Port Number |
| Speed Test IP | Speed Test Server IP Address |
| AMQP IP | AMQP Server IP Address |
| AMQP Port | AMQP Server Port |

## Auto Deployment Flow

The auto deployment script contains below methods as a sequence.

### 

### Modifying configuration Values

To modify the configuration files we are using jinja. All the user inputs will be store in config.yml. Based on the config.yml all the configuration values will be changed with the help of Jinja.

|  |
| --- |
| *# Modify the configuration files*  **def** **modifyConfigs**(self):  *#Load data from YAML into Python dictionary*  config\_data = yaml.load(open('./config.yml'))  *#Load Jinja2 template*  **if** self.doc['MULTI\_NODE']:  env = Environment(loader = FileSystemLoader('./templates/multi\_node'), trim\_blocks=**True**, lstrip\_blocks=**True**)  **else**:  env = Environment(loader = FileSystemLoader('./templates/single\_node'), trim\_blocks=**True**, lstrip\_blocks=**True**)  file\_list = ['api', 'dashboard', 'nodesocket', 'streamprocessor']   **if** self.doc['MULTI\_NODE']:  config\_dir = '/home/versa/deployment\_stack/stacks/multi\_node/configs/'  **else**:  config\_dir = '/home/versa/deployment\_stack/stacks/single\_node/configs/'   **for** f **in** file\_list:  template = env.get\_template(f + SOURCE\_EXT)  *#Render the template with data and print the output*  output\_from\_parsed\_template = template.render(config\_data)  *# to save the results*  **with** open(config\_dir + f + DEST\_EXT, "w") **as** fh:  fh.write(output\_from\_parsed\_template)   self.updateLog('Modified the configuration files') |

### 

### Initiating the Deployment

In this part we will be checking whether it is a single node or multi node. If it is single node (Both SaaS/SaaS+Shop) the docker images will be loaded first and then the stack deployment will be initiated. If it is a multi node the docker images will be loaded in all the three manager nodes first. And the docker swarm init functionality will be started.

|  |
| --- |
| **def** **initiateDeployment**(**self**): |

### Initiate Swarm Cluster

This method only executed for multi node deployment. The swarm node init method was already executed in Initial Deployment method for Multi Node.

So in this method the the script will get the swarm token from the manager0 node. And the join token code will be executed in all other manager nodes (manager1, manager2).

|  |
| --- |
| *# Initiate Swarm Cluster* def swarmCluster(**self**):  **if** **self**.doc['MULTI\_NODE'] == **True**:  dockerSwarmToken = subprocess32.check\_output(['sudo docker swarm join-token manager'], shell=**True**)  dockerSwarmToken = dockerSwarmToken.replace("To add a manager to this swarm, run the following command:", "")  dockerSwarmToken = dockerSwarmToken.strip()  time.sleep(5)  **self**.joinSwarmCluster(dockerSwarmToken, 'NODE\_2\_IP', 'NODE\_2\_USER', 'NODE\_2\_PASSWORD')  time.sleep(5)  **self**.joinSwarmCluster(dockerSwarmToken, 'NODE\_3\_IP', 'NODE\_3\_USER', 'NODE\_3\_PASSWORD')  **else**:  **self**.updateLog("Swarm cluster is not needed") |

After finishing Swarm Cluster, the overlay network will be created for multi node. The below commands will be executed for creating network overlay.

|  |
| --- |
| sudo docker node **update** *--label-add deploy.role=manager0 manager0* sudo docker node **update** *--label-add deploy.role=manager1 manager1* sudo docker node **update** *--label-add deploy.role=manager2 manager2* sudo docker network **create** *--driver overlay --opt encrypted proxy* sudo docker network **create** *--driver overlay --opt encrypted go* sudo docker network **create** *--driver overlay --opt encrypted --subnet=10.16.0.0/16 shop* |

### Configure Mongo

This configure mongo is only needed for Multi Node deployment. And there are two types of mongo deployment is available.

* No Sharding
* Sharding

**No Sharding:** Once deploying mongo replicas (no sharding) we have to run the following command in the mongo container.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mongo\_container\_id** bash -c "echo 'rs.initiate({\_id : \"datars\", members: [{ \_id : 0, host : \"mongodb\_mongo1\", \"priority\"  : 1 },{ \_id : 1, host : \"mongodb\_mongo2\", \"priority\" : 1 },{ \_id : 2, host : \"mongodb\_mongo3\", \"priority\" : 1, \"votes\" : 1 }]})' | mongo" |

The above script will be stored in ***scripts/rsInitiate.sh*** file and the mongo\_container\_id will be replaced with the current mongo container ID. Then the shell script will be executed.

Once finishing the rs initiate command. We have to copy the ***init\_mongo/*** (mongo backup) directory and ***authMongoDB.js*** files to the mongo container.

And now we have to run the following command within mongo container.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mongo\_container\_id** bash -c 'mongo < authMongoDB.js' sudo docker exec -it **mongo\_container\_id** bash -c 'mongorestore --db saas --username versa --password versa123 --authenticationDatabase saas init\_mongo/' |

The above script will be stored in ***scripts/restoreMongo.sh*** file and the mongo\_container\_id will be replaced with the current mongo container ID. Then the shell script will be executed.

**Sharding:** Once deploying mongo cluster (sharding) we have to run the following command in the mongo cfg container for sharding.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mongo\_container\_id** bash -c "echo 'rs.initiate({\_id: \"cfgrs\",configsvr: true, members: [{ \_id : 0, host : \"mongodb\_cfg1\" },{ \_id : 1, host : \"mongodb\_cfg2\" }, { \_id : 2, host : \"mongodb\_cfg3\" }]})' | mongo" |

The above script will be stored in ***scripts/rsInitiateSharding.sh*** file and the mongo\_container\_id will be replaced with the current mongo container ID. Then the shell script will be executed.

Now we are going to build our shard replica set by running the below command.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mongo\_container\_id** bash -c "echo 'rs.initiate({\_id : \"datars\", members: [{ \_id : 0, host : \"mongodb\_data1\", \"priority\" : 1 },{ \_id : 1, host : \"mongodb\_data2\", \"priority\" : 1 },{ \_id : 2, host : \"mongodb\_data3\", \"priority\" : 1, \"votes\" : 1 }]})' | mongo" |

The above script will be stored in ***scripts/rsInitiateShardingData.sh*** file and the mongo\_container\_id will be replaced with the current mongo container ID. Then the shell script will be executed.

Then we have to create routers for sharding. To create routers we need to run the below command in the mongo container.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mongo\_container\_id** bash -c "echo 'sh.addShard(\"datars/mongodb\_data1\")' | mongo" sudo docker exec -it **mongo\_container\_id** bash -c "echo 'sh.addShard(\"datars/mongodb\_data2\")' | mongo" sudo docker exec -it **mongo\_container\_id** bash -c "echo 'sh.addShard(\"datars/mongodb\_data3\")' | mongo" |

The above script will be stored in ***scripts/rsInitiateShardingMongos.sh*** file and the mongo\_container\_id will be replaced with the current mongo container ID. Then the shell script will be executed.

Once finishing the routing, we have to copy the ***init\_mongo/*** (mongo backup) directory and ***authMongoDB.js*** files to the mongo container.

And now we have to run the following command within mongo container.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mongo\_container\_id** bash -c 'mongo < authMongoDB.js' sudo docker exec -it **mongo\_container\_id** bash -c 'mongorestore --db saas --username versa --password versa123 --authenticationDatabase saas init\_mongo/' |

The above script will be stored in ***scripts/restoreMongo.sh*** file and the mongo\_container\_id will be replaced with the current mongo container ID. Then the shell script will be executed.

### 

### Configure Kafka

After completing configuring mongo we have to deploy Kafka.

**Single Node:** For single node, the kafka will be deployed in the initial deployment script. Now we have to create kafka topics. To create the kafka topics we have to run the below command.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 -partitions 1 --topic versa-analytics' sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 -partitions 1 --topic versa-internal' sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 -partitions 1 --topic versa-monitoring' sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 -partitions 1 --topic versa-amqp' sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 -partitions 1 --topic versa-device-backend-update' sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 -partitions 1 --topic versa-update-event' sudo docker exec -it **kafka\_container** bash -c '/opt/kafka/bin/kafka-topics.sh --list --zookeeper zookeeper:2181' |

The above script will be stored in ***scripts/kafkaSingle.sh*** file and the kafka\_container will be replaced with the current kafka container ID. Then the shell script will be executed.

**Multi Node:** For multi node the kafka will be deployed separately. Once completed the stack deployment we have to run the below command in the kafka container. But once the stack is deployed we have to wait for some time. Because the container will take some time to up. To wait for 60 seconds we are using **time**.sleep method of python.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper1:2181 --replication-factor 3 -partitions 6 --topic versa-analytics' sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper1:2181 --replication-factor 3 -partitions 6 --topic versa-internal' sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper1:2181 --replication-factor 3 -partitions 6 --topic versa-monitoring' sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper1:2181 --replication-factor 3 -partitions 6 --topic versa-amqp' sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper1:2181 --replication-factor 3 -partitions 6 --topic versa-device-backend-update' sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --create --zookeeper zookeeper1:2181 --replication-factor 3 -partitions 6 --topic versa-update-event' sudo docker exec -it kafka\_container sh -c '/opt/kafka/bin/kafka-topics.sh --list --zookeeper zookeeper1:2181' |

The above script will be stored in ***scripts/kafkaMulti.sh*** file and the kafka\_container will be replaced with the current kafka container ID. Then the shell script will be executed.

### 

### Configure Mysql

Once Kafka deployment is done we are going to deploy Mysql for Shop. We are having setup.sh file for mysql which contains creating database and importing the database commands. And also it is having a command to create user for haproxy for mysql cluster (Galera)

**Single Node:** For single node the mysql container will be created in the initial deployment. First we need to update App Endpoint and Oauth Endpoint in **init\_mysql/setup.sh** file. After updating the setup.sh file the init\_mysql directory will be copied to mysql container.

After copying init\_mysql directory we need to run the below command in mysql container.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mysql\_container\_id** bash -c './init\_mysql/setup.sh' |

The above script will be stored in ***scripts/restoreMysql.sh*** file and the mysql\_container\_id will be replaced with the current mysql container ID. Then the shell script will be executed.

**Multi Node:** For multi node we have to deploy the mysql stack with mysql\_cluster.yml file.

First we need to update App Endpoint and Oauth Endpoint in **init\_mysql/setup.sh** file. After updating the setup.sh file the init\_mysql directory will be copied to mysql container.

After copying init\_mysql directory we need to run the below command in mysql container.

|  |
| --- |
| **#!/bin/sh** sudo docker exec -it **mysql\_container\_id** bash -c './init\_mysql/setup.sh' |

The above script will be stored in ***scripts/restoreMysql.sh*** file and the mysql\_container\_id will be replaced with the current mysql container ID. Then the shell script will be executed.

### Configure Rsync

Rsync is only needed for Shop + Shop with Multi Node. To prepare Rsync we need to create rsa key in manager0 (where the deployment script is running). To create rsa key we need to run the below command in the terminal.

|  |
| --- |
| **#!/bin/sh** sudo su <<HERE ssh-keygen -t rsa -f ~/.ssh/rsync -q -P "" cat ~/.ssh/rsync.pub > ~/.ssh/authorized\_keys cp ~/.ssh/rsync /home/versa/ cp ~/.ssh/rsync.pub /home/versa/ chmod 777 /home/versa/rsync\* HERE sudo mkdir /opt/versa |

The above script will be stored in ***scripts/keyGen.sh*** file and the shell script will be executed.

To sync all the managers we need to keep the same rsa key. So we need to copy the rsa key file in all the machines with scp module.

The python scp command will look like this.

|  |
| --- |
| scp = SCPClient(ssh.get\_transport()) scp.put('/home/versa/rsync', '/home/versa/') scp.put('/home/versa/rsync.pub', '/home/versa/') scp.put('rsync/keyGenCopy.sh', '/home/versa/keyGenCopy.sh') scp.put('rsync/rsync.conf', '/home/versa/rsync.conf') scp.put('rsync/media\_rsync.sh', '/home/versa/media\_rsync.sh') scp.put('rsync/reseller\_rsync.sh', '/home/versa/reseller\_rsync.sh') scp.put('rsync/static\_cache\_rsync.sh', '/home/versa/static\_cache\_rsync.sh') |

The copied file only be stored in home/versa directory. So we have run one more command to copy the rsa file in /root/.ssh/ directory.

|  |
| --- |
| **#!/bin/sh** sudo su <<HERE mv /home/versa/rsync ~/.ssh/ mv /home/versa/rsync.pub ~/.ssh/ cat ~/.ssh/rsync.pub > ~/.ssh/authorized\_keys chmod 600 ~/.ssh/rsync chmod 600 ~/.ssh/rsync.pub HERE |

The above code will be stored in ***scripts/keyGenCopy.sh*** file and the shell script will be executed in the client machines (Which is manager 1 and manager 2 not manager 0).

After configuring RSA key in all the managers we have to run the below command to configure the Rsync.

|  |
| --- |
| **cd** /home/versa/\**n**" ./keyGenCopy.**sh**\**n**" sudo **mkdir** -p /opt/versa\**n**" sudo sed -i 's/<MANAGER1>/{manager1}/**g**' /home/versa/media\_rsync.**sh**\**n**" sudo sed -i 's/<MANAGER2>/{manager2}/**g**' /home/versa/media\_rsync.**sh**\**n**" sudo sed -i 's/<MANAGER1>/{manager1}/**g**' /home/versa/reseller\_rsync.**sh**\**n**" sudo sed -i 's/<MANAGER2>/{manager2}/**g**' /home/versa/reseller\_rsync.**sh**\**n**" sudo sed -i 's/<MANAGER1>/{manager1}/**g**' /home/versa/static\_cache\_rsync.**sh**\**n**" sudo sed -i 's/<MANAGER2>/{manager2}/**g**' /home/versa/static\_cache\_rsync.**sh**\**n**" sudo mv /home/versa/media\_rsync.**sh** /opt/versa\**n**" sudo mv /home/versa/reseller\_rsync.**sh** /opt/versa\**n**" sudo mv /home/versa/static\_cache\_rsync.**sh** /opt/versa\**n**" sudo mv /home/versa/rsync.**conf** /etc/supervisor/**conf**.**d**/\**n**" sudo service supervisor restart\**n**" |

After configuring rsync in client machines we need to run the below command in the manager 0 (Where the auto script is running)

|  |
| --- |
| cp /home/versa/deployment\_stack/auto\_deploy/rsync/media\_rsync.sh /home/versa/ cp /home/versa/deployment\_stack/auto\_deploy/rsync/reseller\_rsync.sh /home/versa/ cp /home/versa/deployment\_stack/auto\_deploy/rsync/static\_cache\_rsync.sh /home/versa/ cp /home/versa/deployment\_stack/auto\_deploy/rsync/rsync.conf /home/versa/ sudo mkdir -p /opt/versa sudo sed -i 's/<MANAGER1>/{manager1}/g' /home/versa/media\_rsync.sh sudo sed -i 's/<MANAGER2>/{manager2}/g' /home/versa/media\_rsync.sh sudo sed -i 's/<MANAGER1>/{manager1}/g' /home/versa/reseller\_rsync.sh sudo sed -i 's/<MANAGER2>/{manager2}/g' /home/versa/reseller\_rsync.sh sudo sed -i 's/<MANAGER1>/{manager1}/g' /home/versa/static\_cache\_rsync.sh sudo sed -i 's/<MANAGER2>/{manager2}/g' /home/versa/static\_cache\_rsync.sh sudo mv /home/versa/media\_rsync.sh /opt/versa sudo mv /home/versa/reseller\_rsync.sh /opt/versa sudo mv /home/versa/static\_cache\_rsync.sh /opt/versa sudo mv /home/versa/deployment\_stack/auto\_deploy/rsync/rsync.conf /etc/supervisor/conf.d/ sudo service supervisor restart |

After restarting the supervisor in all the managers the rsync setup will be done.

### Deployment

After completing Rsync setup finally the Deployment will be initiated. This step will be containing four types.

* Single Node with SaaS + Shop
* Single Node with SaaS Only
* Multi Node with SaaS + Shop
* Multi Node with SaaS Only

### 

### Configure Logstash

After deployment is finished we need to configure logstash. The logstash should be deployed for both Single Node and Multi Node.

The above script will be run in manager0 (Where the script is running).

### 

### Configure HaProxy

Configuring Haproxy is the final step. Once deployment is done, this step will be initiated. The current script is configured for Versa Web Portal QCOW Image, so it will only work with Versa Webportal QCOW image. To configure we need to run the below command in Versa Webportal Node.

|  |
| --- |
| sudo dpkg -i /home/versa/haproxy/liblua5.3-0\_5.3.3-1\_amd64.deb sudo dpkg -i /home/versa/haproxy/haproxy\_1.6.3-1\_amd64.deb sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg\_bk sudo mkdir /etc/haproxy/certs sudo cp /home/versa/haproxy/cert.pem /etc/haproxy/certs/ sudo cp /home/versa/haproxy/haproxy.cfg /etc/haproxy/ sudo service haproxy restart |

The above script will be executed in the Haproxy Node through ssh.

And there is one more configuration is available. For testing purpose we can use another method to configure Haproxy in Manager 0

That method calling will be commented out. We have to uncomment the following method to configure Haproxy in local machine.

|  |
| --- |
| **deployment**.configureHaproxyLocal() |

To install Haproxy in any Manager 0 node, we need to install the following command.

|  |
| --- |
| **#!/bin/sh** sudo apt-get update && sudo apt-get install haproxy -y sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg\_bk sudo mkdir /etc/haproxy/certs sudo cp /home/versa/deployment\_stack/auto\_deploy/certs/cert.pem /etc/haproxy/certs/ sudo cp /home/versa/deployment\_stack/auto\_deploy/haproxy/haproxy.cfg /etc/haproxy/ sudo service haproxy restart |

The above command will be stored in ***scripts/haproxyLocal.sh*** and it will be executed.

Once after finishing all the steps, the application will be deployed as expected.