THALES

CipherTrust Manager Platform

CipherTrust Manager Tokenization Server (CTS) Deployment Automation

Document Version 1

Contents

PREFACE

Note: In Sept of 2020 Thales has rebranded the KeyManager named KeySecure or KeySecure Next Gen to CipherTrust Manager (CM). It combines capabilities from both the legacy Gemalto KeySecure and the Vormetric Data Security Manager products. Any reference in documentation to KeySecure , NextGen or Data Security Manager can be considered to now be the newly branded CipherTrust Manager (CM) product. See following link for more details:
https://cpl.thalesgroup.com/encryption/ciphertrust-manager

Thales has also rebranded the Vormetric Token Server (VTS) also known as Vormetric Application Crypto Server (VACS) as CipherTrust Tokenization Server (CTS). This product is a vaultless solution and also has been referred to as CipherTrust Token Vaultless (CT-VL)

CipherTrust Manager Tokenization Deployment Automation provides working examples on how to:

1. Automate setup and configuration of CTS assets such as token templates, token groups, users, masks, etc. using REST API's.
2. Automate the creation of the initial VTS node in a Cloud Service Provider such as AWS.
3. Automate the deployment of adding a new node to the CTS cluster. Typically, this is for auto scaling scenarios.

Thales also provides a Tokenization Vaulted solution as well known as CT-V. This document will not cover that product.

DOCUMENTATION VERSION HISTORY

Product/Document Version	Date	Changes
V1.0	5/2021	M. Warner

ASSUMPTIONS

This documentation assumes the reader is familiar with the following Thales products and processes:

- Tokenization
- Key management
- Familiarity with REST

The examples in this document can be used for use cases where either the Legacy Vormetric Data Security Manager or the CipherTrust Manager are used for the Key Manager.

GUIDE TO DOCUMENTATION

Related documents are available to registered users on the Thales Web site at https://cpl.thalesgroup.com/ or https://thalesdocs.com/

SERVICES UPDATES AND SUPPORT INFORMATION

The license agreement that you have entered into to acquire the Thales products (“License Agreement") defines software updates and upgrades, support and services, and governs the terms under which they are provided. Any statements made in this guide or collateral documents that conflict with the definitions or terms in the License Agreement, shall be superseded by the definitions and terms of the License Agreement. Any references made to “upgrades” in this guide or collateral documentation can apply either to a software update or upgrade.

GETTING STARTED

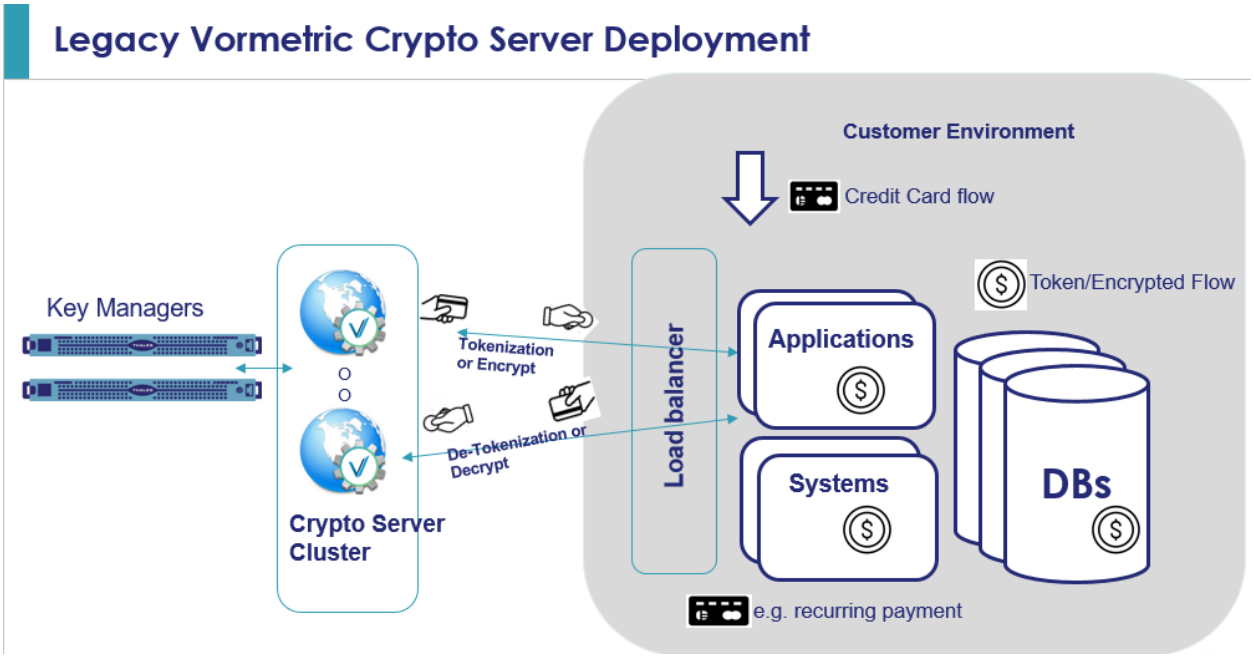
Architecture

CipherTrust Tokenization Server (CTS)

The solution provided in this document is based on having an external key manager to create and manage the encryption keys. This appliance comes in both physical and virtual versions and is call the CipherTrust Manager or the Vormetric Data Security Manager (DSM). Most implementations will have at least two Key Managers handling requests. The Key Manager platform operates as a cluster and it is easy to add more nodes to the cluster if needed.

If the DSM is used as a key manager then both tokenization and encryption capabilities are available using the CTS otherwise the CipherTrust Manager Key Manager can handle REST calls to encrypt and decrypt.

The diagram below is showing the Legacy Vormetric Crypto Server because it shows all the original capabilities of the Vormetric Crypto Server. CTS or (Crypto Server Cluster in the diagram below) is implemented in a cluster as shown below and can easily be scaled up and down by adding more nodes to the cluster.



Note: If the CipherTrust Manager is utilized for encrypt and decrypt REST calls then this must be factored in to the overall design to ensure that the performance impact is planned for and proper

capacity planning can be implemented. The load balancer is not included with the CTS and must be implemented by the customer.

Documentation

https://yourctsnode/vts/admin/v1/doc/swagger/

CTS-2.6.0_InstallAdminProgGuide_v1**.pdf**

Examples

The examples provided in this document can be used for both Vormetric DSM and CipherTrust Manager key manager.

Using the CipherTrust Tokenization Vaultless admin REST API

This sample shows how to use the CTS REST api to create token groups, token templates, masks and users. Please see the two links above in the documentation section for a complete list of endpoints available. This example can be run from any linux machine that can reach the CTS node.

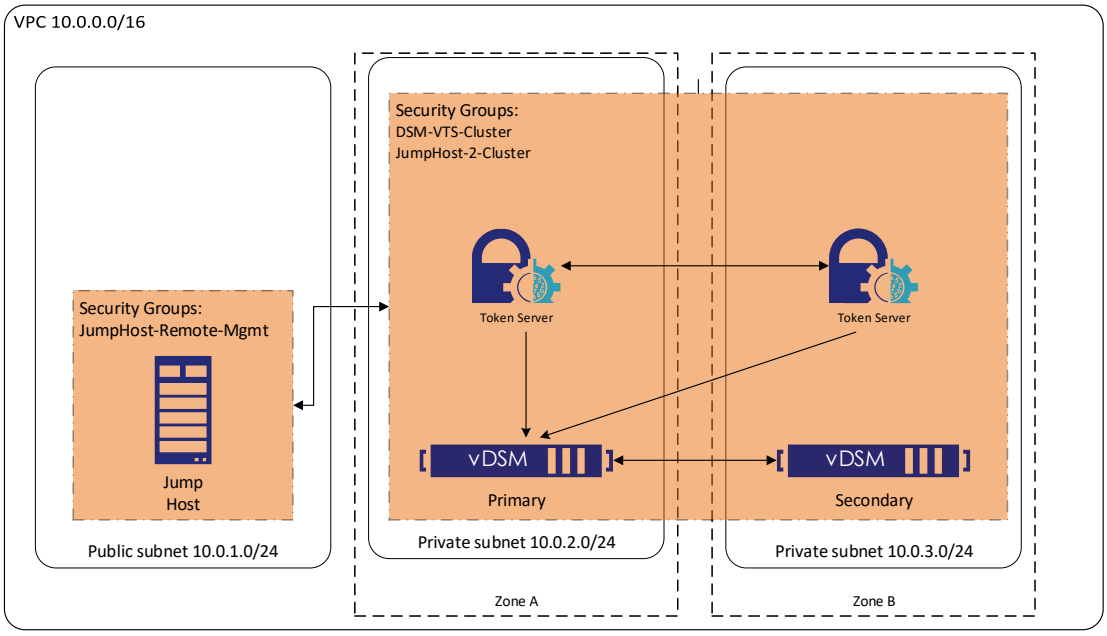
```
#!/bin/bash
CT="Content-Type:application/json"
#This script will create tokengroups,tokentemplates,mask and users in the Token Server (VTS or CTS which is the new branded name).
#For best practices and seperation of duties it assumes you have already created keys in the Key Manager. vts-key-n where n = number of keys.
#After running this script you should then proceed to the VTS UI and assign masks to Users and also to to permissions to assign access to tokenize or detoken for the various keys.
URL='youripadddresstovts'
PWD="youpwd!"
CT="Content-Type:application/json"
Credentials='{"username":"vtsroot","password":"'$PWD'"}'
echo $Credentials
AUTH="curl -k -X POST -H $CT -d $Credentials https://$URL/api/api-token-auth/"
RESPONSE=`$AUTH`
TOKEN=$(echo "$RESPONSE" | jq -r '.token')
echo $TOKEN

# create standard tokengroups/tokentemplates

i=0
for i in {1..1}
do

groupname="vtsgroup"$i
keyname="vts-key-"$i
templatename="vtstemplate"$i
Datagroup='{"name":"'$groupname'","key":"'$keyname'"}'
echo $keyname
echo $groupname
echo $templatename
echo $Datagroup

args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$Datagroup"'' https://$URL/api/tokengroups/)
```

```
RESPONSE2= curl "${args[@]}"
echo $RESPONSE2
Datatemplate='{"name":"'$templatename'","tenant":"'$groupname'" ,"format": "FPE", "keepleft" : 0,
"keepright": 0, "charset": "All digits", "prefix": "", "startyear": null, "endyear": null,
"irreversible": false}'
echo $Datatemplate
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$Datatemplate"' https://$URL/api/tokentemplates/)
RESPONSETemplate= curl "${args[@]}"
echo $RESPONSETemplate

i=$i+1
done

# create project specific tokengroups
Datagroup='{"name":"Demo","key":"vts-key-1"}'
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$Datagroup"' https://$URL/api/tokengroups/)
RESPONSE2= curl "${args[@]}"
echo $RESPONSE2

Datagroup='{"name":"t1","key":"vts-key-1"}'
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$Datagroup"' https://$URL/api/tokengroups/)
RESPONSE2= curl "${args[@]}"
echo $RESPONSE2
# create project specific tokentemplates
cctemplate='{"name":"Credit Card","tenant":"t1" ,"format": "FPE", "keepleft" : 0, "keepright": 0,
"charset": "All digits", "prefix": "", "startyear": null, "endyear": null, "irreversible":
false}'
echo $cctemplate
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$cctemplate"' https://$URL/api/tokentemplates/)
RESPONSEcc= curl "${args[@]}"
echo $RESPONSEcc

numerictemplate='{"name":"Numeric","tenant":"Demo" ,"format": "FPE", "keepleft" : 0, "keepright":
0, "charset": "All digits", "prefix": "", "startyear": null, "endyear": null, "irreversible":
false}'
echo $numerictemplate
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$numerictemplate"' https://$URL/api/tokentemplates/)
RESPONSEnumeric= curl "${args[@]}"
echo $RESPONSEnumeric

texttemplate='{"name":"Text","tenant":"Demo" ,"format": "FPE", "keepleft" : 0, "keepright": 0,
"charset": "Alphanumeric", "prefix": "", "startyear": null, "endyear": null, "irreversible":
false}'
echo $texttemplate
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$texttemplate"' https://$URL/api/tokentemplates/)
RESPONSEtext= curl "${args[@]}"
echo $RESPONSEtext

prefixtemplate='{"name":"prefixexample","tenant":"Demo" ,"format": "FPE", "keepleft" : 0,
"keepright": 0, "charset": "Alphanumeric", "prefix": "pre-", "startyear": null, "endyear": null,
"irreversible": false}'
echo $prefixtemplate
args=(-k -X POST -H 'Authorization: Bearer '"$TOKEN"'' -H Content-Type:application/json -d
'"$prefixtemplate"' https://$URL/api/tokentemplates/)
RESPONSEprefix= curl "${args[@]}"
echo $RESPONSEprefix
#create masks
curl -X POST "https://$URL/api/masks/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"name\": \"showleft6\",
\"showleft\": 6, \"showright\": 0, \"maskchar\": \"?\"}"
```

```
curl -X POST "https://$URL/api/masks/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"name\": \"first2last2\",
\"showleft\": 2, \"showright\": 2, \"maskchar\": \"X\"}"
curl -X POST "https://$URL/api/masks/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"name\": \"all\", \"showleft\":
99, \"showright\": 99, \"maskchar\": \"X\"}"
curl -X POST "https://$URL/api/masks/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"name\": \"last4\",
\"showleft\": 0, \"showright\": 4, \"maskchar\": \"X\"}"



PWD="Customer123!"

#create users

echo "password" $PWD

curl -X POST "https://$URL/api/users/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"username\": \"fraud\",
\"email\":\"fraud@example.com\", \"password\": \"$PWD\", \"is_active\": true, \"is_staff\": true,
\"is_superuser\": false}"

curl -X POST "https://$URL/api/users/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"username\": \"custserv1\",
\"email\":\"custserv1@example.com\", \"password\": \"$PWD\", \"is_active\": true, \"is_staff\":
true, \"is_superuser\": false}"

curl -X POST "https://$URL/api/users/" -H "accept: application/json" -H "Content-Type:
application/json" -H 'authorization: Bearer '"$TOKEN"'' -k -d "{ \"username\": \"custserv2\",
\"email\":\"custserv2@example.com\", \"password\": \"$PWD\", \"is_active\": true, \"is_staff\":
true, \"is_superuser\": false}"
```

Automating the creation of a CTS Cluster in AWS

Listed below is a sample AWS CLI script that creates the security groups, DSM instances, VTS instances and an Ubuntu instance. This can be used as a starting point for project team's implementations and POC's. Here is a diagram of what the script implements.



Note: This example creates a DSM for the Key Manager but the process would be the same for CipherTrust Manager.

```
#!/bin/sh
# AWS DSM instance create script
start_time="$(date -u +%s)"

awsRegionName=us-east-1
jumpRemoteMGMTSGName=JumpHost-Remote-Mgmt-sg
jumpclusterSGName=JumpHost-2-Cluster-sg
dsmvtsSGName=DSM-VTS-Cluster-sg
jumpHostAMIName=ami-059eeca93cf09eebd
awstag=mytagvtssplit
awsAZ1=us-east-1a
awsAZ2=us-east-1b
jumphostsubnet=subnet-f3aab8ae

aws configure set default.region $awsRegionName
#find our vpc
vpcID=$(aws ec2 describe-vpcs --query 'Vpcs[*][VpcId]' --output text |
sort -r | head -1)
echo "vpcid=$vpcID"
#could have mutliple dsm ami's. find most recent one.
dsmAMI=$(aws ec2 describe-images --owners self --query
'Images[*][CreationDate , Name,ImageId]' --output text | grep dsm | sort -
r | head -1 | awk '{print $3}')
echo "dsmami=$dsmAMI"
#could have mutliple vts ami's. find most recent one.
vtsAMI=$(aws ec2 describe-images --owners self --query
'Images[*][CreationDate , Name,ImageId]' --output text | grep vts | sort -
r | head -1 | awk '{print $3}')
echo "vtsAMI=$vtsAMI"
#Create the jumpRemoteMGMTSG security groups
jumpRemoteMGMTSG=$(aws ec2 create-security-group --group-name
$jumpRemoteMGMTSGName --description "jumpHost-Remote-Mgmt-sg" --vpc-id
$vpcID --output text)
echo "jumpRemoteMGMTSG=$jumpRemoteMGMTSG"
#Create the jumpclusterSG security groups
jumpclusterSG=$(aws ec2 create-security-group --group-name
$jumpclusterSGName --description "JumpHost-2-Cluster-sg" --vpc-id $vpcID -
-output text)
echo "jumpclusterSG=$jumpclusterSG"
#Create the DSM and VTS security groups
dsmvtsSG=$(aws ec2 create-security-group --group-name $dsmvtsSGName --
description "DSM-VTS-Cluster-sg" --vpc-id $vpcID --output text)
echo "dsmvtsSG=$dsmvtsSG"

#ip=$(curl https://checkip.amazonaws.com/)
#cidr="$ip/24"
cidr="0.0.0.0/0"
echo "using cidr of $cidr"
echo "Setting jumpRemoteMGMTSG rules"
aws ec2 authorize-security-group-ingress --group-id $jumpRemoteMGMTSG --
protocol tcp --port 22 --cidr $cidr
aws ec2 authorize-security-group-ingress --group-id $jumpRemoteMGMTSG --
protocol tcp --port 3389 --cidr $cidr

echo "Setting jumpclusterSG rules"
aws ec2 authorize-security-group-ingress --group-id $jumpclusterSG --
protocol tcp --port 22 --source-group $jumpRemoteMGMTSG
aws ec2 authorize-security-group-ingress --group-id $jumpclusterSG --
protocol tcp --port 443 --source-group $jumpRemoteMGMTSG
```

```
#aws ec2 authorize-security-group-ingress --group-id $jumpclusterSG --
protocol icmp --port 0-65535 --source-group $jumpRemoteMGMTSG

echo "Setting DSM inbound rules"
aws ec2 authorize-security-group-ingress --group-id $dsmvtsSG --protocol
tcp --port 0-65535 --source-group $dsmvtsSG
#aws ec2 authorize-security-group-ingress --group-id $dsmvtsSG --protocol
icmp --port 0-65535 --source-group $dsmvtsSG --cidr $cidr

awsSubnet1=$(aws ec2 create-subnet --vpc-id $vpcID --availability-zone
$awsAZ1 --cidr-block 172.31.96.0/20  | grep SubnetId | awk '{print $2}' |
tr -d '"' | tr -d ',')
echo "awsSubnet1=$awsSubnet1"
awsSubnet2=$(aws ec2 create-subnet --vpc-id $vpcID --availability-zone
$awsAZ2 --cidr-block 172.31.112.0/20  | grep SubnetId | awk '{print $2}' |
tr -d '"' | tr -d ',')
echo "awsSubnet2=$awsSubnet2"

echo "Creating DSM Instance "
#Create DSM instance.  Need the subnet so VTS and Agent can be in same
network.
dsmInstanceId=$(aws ec2 run-instances --image-id $dsmAMI --count 1 --
instance-type m4.large --security-group-ids $dsmvtsSG $jumpclusterSG --
subnet-id $awsSubnet1 | grep InstanceId | awk '{print $2}' | tr -d '"' |
tr -d ',')
if [ $? -ne 0 ]; then
   echo "Failed to Create DSM instance"
   exit 1
fi
echo "dsmInstanceId=$dsmInstanceId"

echo "Creating VTS Instance "
vtsInstanceId=$(aws ec2 run-instances --image-id $vtsAMI --count 1 --
instance-type t2.xlarge --security-group-ids $dsmvtsSG $jumpclusterSG  --
subnet-id $awsSubnet1  |  grep InstanceId | awk '{print $2}' | tr -d '"' |
tr -d ',')
if [ $? -ne 0 ]; then
   echo "Failed to Create VTS instance"
   exit 1
fi
echo "vtsInstanceId=$vtsInstanceId"

echo "Creating VTE Ubuntu instance "
jump1InstanceId=$(aws ec2 run-instances --image-id $jumpHostAMIName --
count 1 --instance-type t2.micro --key-name MyKeyPair --security-group-ids
$jumpRemoteMGMTSG --subnet-id $jumphostsubnet | grep InstanceId | awk
'{print $2}' | tr -d '"' | tr -d ',')
if [ $? -ne 0 ]; then
   echo "Failed to Create $jumpHostAMIName instance"
   exit 1
fi

#second DSM and VTS in different AZ
echo "Creating DSM Instance "
#Create DSM instance.  Need the subnet so VTS and Agent can be in same
network.
dsmInstanceId2=$(aws ec2 run-instances --image-id $dsmAMI --count 1 --
instance-type m4.large --security-group-ids $dsmvtsSG $jumpclusterSG --
subnet-id $awsSubnet2 | grep InstanceId | awk '{print $2}' | tr -d '"' |
tr -d ',')
if [ $? -ne 0 ]; then
   echo "Failed to Create DSM instance"
   exit 1
fi
echo "dsmInstanceId2=$dsmInstanceId2"
```

```
#dsmSubnet=$(aws ec2 describe-instances --instance-ids $dsmInstanceId --
query 'Reservations[0].Instances[0].SubnetId' | tr -d '"')
echo "Creating VTS Instance "
vtsInstanceId2=$(aws ec2 run-instances --image-id $vtsAMI --count 1 --
instance-type t2.xlarge --security-group-ids $dsmvtsSG $jumpclusterSG  --
subnet-id  $awsSubnet2 |  grep InstanceId | awk '{print $2}' | tr -d '"' |
tr -d ',')
if [ $? -ne 0 ]; then
   echo "Failed to Create VTS instance"
   exit 1
fi
echo "vtsInstanceId2=$vtsInstanceId2"

aws ec2 create-tags --resources $dsmInstanceId $vtsInstanceId
$dsmInstanceId2 $vtsInstanceId2 $jump1InstanceId  --tags
Key=owner,Value=$awstag
cat <<EOF
Summary of Vormetric Run
DSM instance ID: $dsmInstanceId
DSM instance ID2: $dsmInstanceId2
VTS instance ID: $vtsInstanceId
VTS instance ID2: $vtsInstanceId2
JumpHost instance ID: $jump1InstanceId
EOF

end_time="$(date -u +%s)"
elapsed="$(($end_time-$start_time))"
echo "Total of $elapsed seconds elapsed for process"
echo "Done "
exit 0
```

Automating adding a new node to the CTS Cluster

This example automates adding a new node to the CTS cluster. It requires the sshpass linux utility and was tested with CTS 2.6. This example shows integration with the CipherTrust Manager as a key manager. Commands for DSM would be slightly different for step 1 listed below. The CTS installation guide provides details for both Key Manager setups. This example leverages the new "Automated CTS cluster setup", which allows for the remotejoin using the apiadmin user in step 2 below. More details can be found in the CTS-2.6.0_InstallAdminProgGuide_v1.pdf document on the Thales support site. This example can be run from any linux machine that can reach both the CTS existing node and the new node to be added to the cluster.

```
#setenv.sh

export cmip=192.168.159.160
export vtsprimarynode=192.168.159.141
export vtsport=9005
export vtscmuser=cmadmin
export cmadminuser=admin
export APIADMIN_PW=Yoursupersecret123!
export vtsnewnodename=vtsnode2
export vtsnewnodeip=192.168.159.142
export vtsseedkey=vts-seed-key

#1-configure_new_vts_node.sh
echo system timezone --set America/New_York | sshpass -e ssh cliadmin@$vtsnewnodeip
echo system hostname --set $vtsnewnodename | sshpass -e ssh cliadmin@$vtsnewnodeip
echo icapi set --IP $cmip --port $vtsport --user $vtscmuser | sshpass -e ssh cliadmin@$vtsnewnodeip
echo icapi register --host $newvtsip --user $cmadminuser | sshpass -e ssh cliadmin@$vtsnewnodeip
echo cluster apiadmin --setpassword $APIADMIN_PW | sshpass -e ssh cliadmin@$vtsnewnodeip
echo cluster apiadmin --setpassword $APIADMIN_PW | sshpass -e ssh cliadmin@$vtsprimarynode

#status.sh
echo cluster show --nodeip | sshpass -e ssh cliadmin@$vtsnewnodeip
echo cluster show --status | sshpass -e ssh cliadmin@$vtsnewnodeip

# 2-cluster_setup_new_vts_node.sh
echo icapi test --server_connection | sshpass -e ssh cliadmin@$vtsnewnodeip
echo cluster remotejoin $vtsnewnodeip | sshpass -e ssh cliadmin@$vtsprimarynode
echo cluster adpiadmin --diable | sshpass -e ssh cliadmin@$vtsprimarynode
echo cluster adpiadmin --diable | sshpass -e ssh cliadmin@$vtsnewnodeip
```