Boomi Cloud™ API Management - Local Edition

Cluster Design Guide

Version 5.6.2 | November 2024

Contents

Pod - Level Sizing

Cluster sizing recommends number of pods of each type that are required for different Queries Per Seconds (QPS).

Pod or Container Sizing

Pod sizing covers two aspects- resources required by a pod and number of pods required for a given QPS and number of refresh token requests for OAuth.

The sizing guidelines are generic and resource requirements will vary by factors like average payload per traffic request and response, size of config, and number of oauth (refresh + create tokens) requests.

Because of the upstream feature of td-agent-bit; number of log pods are nearer to the number of tm pods.

Pod Characteristics

Pod Type	Memory Usage	CPU Usage	Storage Usage	Network
TML-NoSQL	Normal - High (in case of large Oauth traffic)	Normal	High in case of large oauth traffic	High due to registry activity
TML-CM	High (host 4 services)	Normal	Low	Low
TML-LOG	High	High	Very high	very high
TML-SQL	Normal	Normal	Normal	Normal
TML-Cache	High	Normal	low	High (depending on traffic calls)
TML-TM	Normal (will see a lot of	High	very low	High

Pod Type	Memory Usage	CPU Usage	Storage Usage	Network
	G1GC young gc activity)			
TML-Reporting	Hiigh	High	Very High	High

Limits and Requests

Requests are initial allocation of resources and limits define the max memory or CPU a pod can utilise.

We can define limits for CPU or CPU time, and memory required. When defining limits a general recommendation is to set the value for requests to half of limit. A general rule of thumb is to allow a 20% overhead or bump up space for memory and CPU and to ensure that the application is not paging.

By defining resource limits, you have the following benefits:

- Pods and containers consume resources and there will be situations where one pod can consume more resources leaving other pods starved for them; a starved pod will be restarted.
- Memory leaks in the application will drain nodes of memory
- Optimizes use of resources instead of over-provisioning
- Allows for automatic horizontal scaling of Cache, Log and traffic manager pods

For more information about units of resource:

- For kubernetes, see Resource Units in Kubernetes
- For docker, see Runtime options with Memory, CPUs, and GPUs

Kubernetes Pod Sizing

Caution:

1. The following data captured is for indicative recommendations only.
2. You can choose to vertically scale by providing higher CPU and memory requests and limits.

Idle state utilization observed

Service	Number of Pods	Expected Utilization per Pod	
		Memory	CPU
Cluster manager	1	1GB	500m/0.5 cpu time
SQL	1	1GB	100m/0.1 cpu time
NoSQL	1	2GB	500m/0.5 cpu time
Cache	1	500MB	100m/0.1 cpu time
Log	1	1GB	500m/0.5 cpu time
TM	1	500MB	500m/0.5 cpu time
Reporting	1	1GB	100m/0.1 cpu time

500 QPS

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
		Memory	CPU	Request	Limit	Request	Limit
Cluster manager	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
SQL	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
NoSQL	1	1 GB	1200m/1.2 cpu time	1000Mi	1250mi	1000m	1500m
Cache	1	500MB	500m/0.5 cpu time	500mi	625Mi	500m	625m
Log	1	2GB	1200m/1.2 cpu time	1000Mi	2500mi	1000m	1500m
TM	1	1.5GB	1500m/1.5 cpu time	1000Mi	1900mi	1000m	1900m
Reporting	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m

1500 QPS In this scenario 50% of traffic is protected by OAuth and an additional 750 QPS is for refresh tokens

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
		Memory	CPU	Request	Limit	Request	Limit
Cluster manager	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
SQL	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
NoSQL	3	1 GB	1200m/1.2 cpu time	1000Mi	1250mi	1000m	1500m
Cache	2	1 GB	500m/0.5	1000Mi	1250mi	500m	625m

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
			cpu time				
Log	4	2GB	1200m/1.2 cpu time	1000Mi	2500mi	1000m	1500m
TM	5	1.2GB	1200m/1.2 cpu time	1000Mi	1500mi	1000m	1500m
Reporting	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m

3000 QPS In this scenario 50% of traffic is protected by OAuth and an additional 1500 QPS is for refresh tokens

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
		Memory	CPU	Request	Limit	Request	Limit
Cluster manager	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
SQL	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
NoSQL	3	1 GB	1200m/1.2 cpu time	1000Mi	1250mi	1000m	1500m
Cache	2	1 GB	500m/0.5 cpu time	500Mi	1250mi	500m	625m
Log	8	2.2GB	1200m/1.2 cpu time	1000Mi	2750mi	1000m	1500m
TM	10	1.2GB	1200m/1.2 cpu time	1000Mi	1500mi	1000m	1500m

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
Reporting	1	1.5GB	3000m/3 cpu time	1000Mi	1900Mi	500m	3750m

3000 QPS with TM vertically scaled and varied payload sizes

> ⚠ ***Caution:***
>
> - *All traffic was open i.e. not authenticated via Oauth but controlled.*
> - *Request and response payload was randomly varied between 1Kb to 1Mb*
> - *Traffic manager pods were provided with 2000 mi cpu time instead of 1*
> - *Log pods required higher memory as disk writes were not scaled with the larger payloads and RAM was used to hold log events waiting to be written*

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
		Memory	CPU	Request	Limit	Request	Limit
Cluster manager	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
SQL	1	1GB	500m/0.5 cpu time	1000Mi	1250Mi	500m	625m
NoSQL	3	1 GB	1200m/1.2 cpu time	1000Mi	1250mi	1000m	1500m
Cache	2	1 GB	500m/0.5 cpu time	500Mi	1250mi	500m	625m
Log	8	3GB	1200m/1.2 cpu time	1000Mi	3750mi	1000m	1500m
TM	10	1.2GB	1200m/1.2 cpu time	1000Mi	1500mi	1000m	2500m

Service	Number of Pods	Observed Utilization Per Pod		Memory Per Pod		CPU Per Pod	
Reporting	1	1.5GB	3000m/3 cpu time	1000Mi	1900Mi	500m	3750m

> **Caution:** Pod performance is dependent on factors like network in most cases and storage speed especially for log pods. Therefore provide higher resources for pods based on their characteristics.

Requirements for different QPS shown here are based on actual observation and the number of pods for TM, Cache and Log have been determined by the following HPA rules:

```
kubectl autoscale deployment tm-deploy-0 --min=1 --max=5
--cpu-percent=80
kubectl autoscale statefulsets log-set-0 --min=1 --max=5
--cpu-percent=80
kubectl autoscale statefulsets cache-set-0 --min=1 --max=5
--cpu-percent=80
```

Pod Placement on Nodes

In the case where users would not like to request and set resource limits, the resource usage characteristics of each pod dictate placement of workloads on nodes. High availability deployment also dictates that pods of same type don't end up on same node.

Pod/Node affinity	Local Edition-NoSQL	Local Edition-Cm	Local Edition-Log	Local Edition-SQL	Local Edition-Cache	Local Edition-TM	Local Edition-Reporting
Local Edition-NoSQL	⛔	✅	⛔	⚠️ as long as the NoSQL instance is not a seed node	⛔	⛔	⛔
Local Edition-Cm	✅	N/A	✅	✅	✅	✅	✅
Local Edition-LOG	⛔	⛔	⛔	⛔	⛔	⛔	⛔
Local Edition-SQL	⚠️ as long as the NoSQL instance is not a seed node	✅	⛔	⛔	✅	⛔	⛔
Local Edition-Cache	⛔ (both are network intensive)	✅	⛔	✅	⛔	✅	⛔
Local Edition-TM	⛔	⛔	⛔	⛔	✅	⛔	⛔
Local Edition-Reporting	❌ use dedicated node	❌ use dedicated node	❌ use dedicated node	❌ use dedicated node	❌ use dedicated node	❌ use dedicated node	N/A

Network Considerations

Local Edition is compatible with all CNCF certified CNIs.

Make sure the POD network is initialized with a unique set of CIDR. Services should be properly deployed with a unique service IP for POD-POD communication

Storage Considerations

To enable workload to be shifted to other nodes and to allow addtion of new pods, use dynamic provisioning applying storage provisioners instead of manually creating persistent volumes.

Node - Level Sizing

This section represents cluster sizing recommendations based on various tests performed on Local Edition 5.x. Some tests used default deployments scripts shipped with Local Edition, while other tests used customized deployment files (not part of default scripts). Customizations included running only Traffic Managers on dedicated nodes in a cluster, while other components were on remaining nodes in the same cluster. The following tables can be referenced as a guideline for creating K8S clusters as per your high performance requirements, expressed as Transactions per Second (TPS), etc. These tests were performed on Local Edition 5.x with variations in response size and latency of the backend.

Test - Part 1

Test environments details:

1. All the nodes in each cluster were 4 core CPU and 15 GB memory.
2. Load Generation: 8 Jmeter hosts in US West1b (GCP)
3. Backend: 8 Latency Injector hosts in US West1b
4. Local Edition cluster Region: US Central1 (GCP).
5. These tests were performed when TMs used getStats method on memcache servers for time synchronization(default behavior, as of Local Edition 5.3.1).

Response Size (Latency) → Cluster Type ↓	2b (0 ms)	1kb (100 ms)	256kb (500 ms)	1-64kb (100- 300 ms)	1-8kb (30-180 ms)	4-128kb (100-300 ms)
	Unprotected | Protected (OAuth)					
Xtra Small	734 | 781	723 | 571	265 | 229	688 | 610	761 | 545	647 | 385
Small-1	1920 |	1600 |	692 |	1470 | 1200	1850 |	1420 | 1300

Response Size (Latency) → Cluster Type ↓	2b (0 ms)	1kb (100 ms)	256kb (500 ms)	1-64kb (100- 300 ms)	1-8kb (30-180 ms)	4-128kb (100-300 ms)
	Unprotected | Protected (OAuth)					
	1800	1460	590		1320	
Small-2	1300 | 1230	1270 | 1100	366 | 283	1000 | 913	1330 | 733	1000 | 723
Medium-1	2300 | 1600	2100 | 1900	909 | 683	1870 | 1470	2100 | 2000	1950 | 1850
Medium-2	4400 | 3300	3500 | 3300	1300 | 12	3800 | 3400	4200 | 2500	3500 | 3000
Large-1	2300 | 2200	1800 | 1470	1420 | 1100	2500 | 2300	2300 | 1700	1750 | 1700
Large-2	4000 | 3700	3850 | 3300	1500 | 1400	3500 | 3000	4000 | 3380	3000 | 2900

Description of the Topology

Topology	Description	NoSQL Count	Configuration Manager Count	Log Count	SQL Count	Cache Count	TM Count
Xtra Small	No of K8S worker Node -1	1	1	1	1	1	1
Small	No of K8S worker Node -2	1	1	1	1	2	3
Small-2	No of K8S worker Node - 2	1	1	1	1	1	1

Topology	Description	NoSQL Count	Configuration Manager Count	Log Count	SQL Count	Cache Count	TM Count
	One Node dedicated for Traffic manager while remaining containers running on another node.						
Medium-1	No of K8S worker Node - 3	3 (1 per node)	1	2 (max 1 per node)	1	3 (1 per node)	10 (max 4 per node)
Medium-2	Same as Medium-1. But each node has double the capacity, i.e. 8 core and 30 GB	3 (1 per node)	1	2 (max 1 per node)	1	3 (1 per node)	10 (max 4 per node)
Medium-3	Similar to medium-2 cluster of 3 nodes with 3 TMs (same as medium-1), but each node is 2 core and	3 (1 per node)	1	2 (max 1 per node)	1	3 (1 per node)	3 (max 4 per node)

Topology	Description	NoSQL Count	Configuration Manager Count	Log Count	SQL Count	Cache Count	TM Count
	8GB. This test has been done to get no for licensing for total of 6 core.						
Large-1	No of K8S worker Node - 5	3 (max 1 er node)	1	5 (max 1 per node)	1	3 (max 1 per node)	20 (max 4 per node)
Large-2	No of K8S worker Node - 6 3 nodes dedicated to all the 15 TMs. All remaining components running on remaining 3 nodes.	3 (max 1 per node)	1	2 (max 1 per node)	1	3 (max 1 per node)	15 (max 5 per node)

Test - Part 2

In the first part of the test, a limit of 20-21K TPS was reached for the extra large cluster. TPS was not increasing linearly as the cluster scaled horizontally. On further analysis, it was determined that for each request, each TM was connecting to each connected memcache server to get time reference for quota enforcement which created a bottleneck. Better throughput is available if the system time of TMs is used and by

making sure that all the servers are in sync (using NTP or through other mechanism). The property to switch between these two mechanisms is available in TM but it is not exposed via TM property file. This is achieved by using the tml_tm_properties.json deployment property file. To use the cache servers as a shared time reference, set the use_system_time property value to false.

Create a k8s secret out of this property file and overwrite the existing template inside the TM container. With this tweak, a TPS of 110K-115K was achieved for a specific cluster. In this series of tests, extra large type of clusters were created with varying nodes from 8 to 30. In each of the cluster, TMs were run separately on dedicated nodes, log containers on dedicated nodes and other type of containers shared the remaining nodes. No two nosql or cache or log or TMs were running on the same node using K8S anti-affinity feature. CPU utilization was around 55-60% on TM nodes when reaching these max TPS. The details of the tests are shown in the following table.

Cluster	TPS (Unprotected)
Extra Large -1 10 Nodes. 5 TMs - each on a single node dedicated for TM 2 Logs - each on single node dedicated for log 3 NoSQL - each on separate node 3 Cache - each on separate node but shared with NoSQL 1 SQL - On a node shared with NoSQL/cache 1 CM - On a node shared with NoSQL/cache	40000
Extra Large -2 15 Nodes. 8 TMs - each on a single node dedicated for TM 2 Logs - each on single node dedicated for log 3 NoSQL - each on separate node 3 Cache - each on separate node but shared with NoSQL	55000

Cluster	TPS (Unprotected)
1 SQL - On a node shared with NoSQL/cache 1 CM - On a node shared with NoSQL/cache	
Extra Large -3 20 Nodes. 13 TMs - each on a single node dedicated for TM 2 Logs - each on single node dedicated for log 3 NoSQL - each on separate node 5 Cache - each on separate node but shared with NoSQL 1 SQL - On a node shared with NoSQL/cache 1 CM - On a node shared with NoSQL/cache	85,000
Extra Large -4 27 Nodes. 20 TMs - each on a single node dedicated for TM 2 Logs - each on single node dedicated for log 3 NoSQL - each on separate node 5 Cache - each on separate node but shared with NoSQL 1 SQL - On a node shared with NoSQL/cache 1 CM - On a node shared with NoSQL/cache	110,000

The deployment files were customized so that each TM and log was running on a single and separate node while NoSQL, SQL, and CM caches shared the separate nodes. However, while sharing nodes, each NoSQL and cache were running on separate nodes. The pod's anti affinity rules were combined with node label. Each cluster was divided into in three groups. One group of nodes were labeled with label (deploy = tm) where only TMs were deployed, the second group of nodes were labeled as (deploy = log) where only log was deployed while the third group of nodes were labeled as (deploy = other) for running remaining components. We used these labels in corresponding

deployment files under the **nodeSelector** attribute. Contact Local Edition Support for the customized deployment file. This is a compressed system folder usually found in the deployment folder, which has updated or customized yaml files.

Liveness for Boomi Cloud™ API Management - Local Edition Components

Scaling up or scaling down of pods or containers is required in burst and trickle down situations to ensure that the services are resilient.

It is a good practice to have odd number of NoSQL pods per zone and odd number of zones for tolerance and resilience. Cache, Log, and TM pods can scale by setting HPA rules in Kubernetes and Openshift but has to be manually done in Docker swarm clusters

The kubelet uses liveness probes to know when to restart a container. For example, liveness probes could catch a deadlock, where an application is running, but unable to make progress. Restarting a container in such a state can help to make the application more available. In case of Boomi Cloud™ API Management - Local Edition(Local Edition), liveness can be defined for NoSQL, CM and SQL container. Liveness criteria for these containers is that services present inside these containers should be running.

Liveness can be defined only for k8s and openshift setup not for docker swarm setup

Suggested Parameters For Liveness

Container	Command	Initial Delay Seconds	Period Seconds	Failure Threshold	Success Threshold	Log File
CM	/usr/local/bin/cm-liveness-probe.sh	800	30	12	1	cm_liveness_probe.log
NoSQL	/usr/local/bin/nosql-liveness-probe.sh	500	30	10	1	nosql_liveness_probe.log

Container	Command	Initial Delay Seconds	Period Seconds	Failure Threshold	Success Threshold	Log File
SQL	/usr/local/bin/sql-liveness-probe.sh	300	30	10	1	sql_liveness_probe.log

Note: These suggested parameters depends on number of containers, installation platform, network connectivity and so on. In case container is restarting itself again and again while cluster set up due to liveness probe failure, these suggested parameters are to be adjusted.

Liveness Logs

Liveness log file in the container can be found at location - /opt/mashery/containeragent/log. For every day new file is generated for liveness logs. Liveness logs more than 5 days ago are deleted from container.

Using Sample Scripts

Sample liveness scripts for CM, NoSQL and SQL containers are present in the Local Edition distribution package at location samples/deploy/onprem/k8s

Procedure

1. From sample liveness script, copy the following livenessProbe section defined under containers section. Do not copy complete sample script.

```
livenessProbe:
  exec:
    command:
    - /bin/bash
```

```
  - -c
  - /usr/local/bin/sql-liveness-probe.sh
failureThreshold: 10
initialDelaySeconds: 300
periodSeconds: 30
successThreshold: 1
```

2. Put the liveness probe section in the container yaml file under manifest folder.

   For example - liveness probe for SQL container should be put in all sql-pod-*.yaml present under manifest folder.

3. In the yaml file, liveness probe section should be present under containers section after the imagePullPolicy tag.

   Indentation of liveness probe section should be proper. It should be in line with the imagePullPolicy tag.

Cluster Storage Summary

This section describes storage sizing for different components.

Log

The Log container receives various logs from all other containers. The following information summarizes the disk requirements for a single log pod/ container assuming one instance of each all other containers.

Some processes, such as container agent or metrics collector, run inside each container and generate logs at an almost fixed rate given no traffic. The variation comes from the Traffic Manager container as it will generate access logs at a different rate at different QPS and from NoSQL pod when a lot of tokens are being created frequently. But on average, the assumption is that the disk size for a Log container can be computed as sum of size for logs of constant processes of other containers and size of access logs which can vary according to QPS.

Disk size for one log pod/ container for 1 day = Disk size for log from all containers (1 instance of each type) for 1 day + Disk Size for access log generated by Traffic Manager.

Size of logs generated by one instance of each component in one day:

- NoSQL - 65 MB
- SQL - 60 MB
- Cache - 65 MB
- CM - 95 MB
- TM (excluding Access Log) - 65 MB
- Log - 40 MB

So the disk required (approximate) for one day could be calculated as:

Total disk (in MB) = (No of NoSql * 65)+ (No of Sql * 60) + (No of Cache * 65) + (No of CM * 95) + (No of TM * 65) + (No of Log * 40) + (access log size for a given QPS)

For example, if you are running one instance of each component and average QPS is around 200, the disk size required for a single day would be:

disk = (1 * 65) + (1 * 60) + (1 * 65) + (1 * 95) + (1 * 65) + (1 * 40) + (360 * 2 * 24) = 17670 MB (17.67 GB)

Note: 360 MB is the size of Access Log generated at 200 QPS in 30 minutes. Refer to the following table on Unprotected Traffic.

Test Duration	**30 Minutes**											
	container	210 QPS			406 QPS			602 QPS			788 QPS		
		before	**after**	**delta (Mb)**	**before**	**after**	**delta (Mb)**	**before**	**after**	**delta (Mb)**	**before**	**after**	**delta (Mb)**
/mnt/data/trafficmanager/	log-set-0-0	0.46	0.82	**0.36**	0.82	1.60	**0.78**	1.60	2.50	**0.90**	2.50	3.80	**1.30**
/mnt/data/trafficmanager/access/		0.21	0.37	**0.16**	0.37	0.67	**0.30**	0.67	1.20	**0.53**	1.20	1.70	**0.50**
/mnt/data/trafficmanager/enriched/		0.25	0.45	**0.20**	0.45	0.83	**0.38**	0.83	1.40	**0.57**	1.40	2.10	**0.70**

NoSQL

NoSQL storage requirement is primarily driven by number of OAuth tokens needed. For example, if the tokens are getting created at the rate of 200 QPS, it will need close to 177 MB space. Refer to the following table on OAuth (Token Creation).

Test Duration	**30 Minutes**											
	container	200 QPS			354 QPS			590 QPS			757 QPS	
		before	**after**	**delta (Mb)**	**before**	**after**	**delta (Mb)**	**before**	**after**	**delta (Mb)**	**before**	**after**	**delta (Mb)**
Number of Tokens	cass-set-0-0	179,319	394,787	**215,468**	394,787	727,040	**332,253**	727,040	1,215,676	**488,636**	1,215,676	1,559,298	**343,622**
/var/lib/cassandra		87	264	**177**	264	575	**311**	575	1126	**551**	1126	1843	**717**
/var/lib/cassandra/commitlog/		48	137	**89**	137	301	**164**	301	573	**272**	573	922	**349**
/var/lib/cassandra/data/		39	128	**89**	128	275	**147**	275	510	**235**	510	828	**318**

Cache

Cache component does not require much storage, so you can select as minimal as possible.

SQL

SQL database primarily stores configuration data. 2 GB storage should suffice.

> **Note:** The above calculation is based on some assumptions. Therefore, you should consider some buffer while sizing.

High Availability Cluster Design

Local Edition architecture is quite flexible, and can be scaled up or down as needed. But other than TPS consideration, to achieve High Availability (HA), you need to carefully design the cluster. Container orchestrators (K8S or Swarm) can take care of components failure, for example, if one Local Edition component shuts down due to some reason, they will bring another one to maintain the given number of instances if all the criteria are satisfied. But to achieve HA in case of infrastructure failure (such as node failure, zone failure, etc.) of K8S or Swarm, extra planning should be done before creating cluster. HA at different levels requires different planning. The following sections provide some general guidelines to achieve HA for the Local Edition cluster to work as expected.

Cluster considerations for High Availability

Node/Instance redundancy

If your K8S or swarm cluster has been designed just to meet a TPS requirement, then node failures might degrade Local Edition functioning. Situations can be more challenging in case you placed some deployment constraints during initial deployment without HA considerations. One such deployment constraint could be that three NoSQL pods should be deployed on three different nodes in a three K8S worker nodes cluster. If any node failure happens in this scenario, then the NoSQL pod which was running on this node will not be redeployed on remaining two nodes. Unless the third node joins the K8S cluster, the Local Edition cluster will work (assuming no other constraint) but remain in inconsistent state. Boomi recommends having extra node/s in the K8S/ Swarm cluster if any pod/ container deployment constraints is in place. Even in the case of no deployment constraint, extra nodes might come handy in maintaining TPS in case of node failure.

Availability Zone Redundancy

Local Edition supports multi zone deployment. You can deploy Local Edition components spread across different availability zones in all major cloud platforms (AWS, Azure, GCP, etc.). In multi zone deployment, configuration and token data is continuously synced across zones, so even in case of a zone failure, another zone can still serve the traffic. Multi zone deployment should be considered during the initial planning phase itself. Extending the existing cluster is currently not supported. Also, out

of the box multi zone deployment is supported only in K8S. But it can be achieved in Docker Swarm as well.

Relative Sizing

The following section describes Local Edition relative (T-Shirt) sizing topologies and corresponding configurations.

Note: The value for k8s_node_size varies according to cloud environment. **t2.xlarge** is for 4 core CPU and 16 GB memory machines in AWS. The equivalent machine would be **n1-standard-4** in GCP and **D4s_v3_standard** in Azure.

Topology	Kubernetes Configuration	Pod Configuration
Xtra Small	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 1 k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	tml_cm_count: 1, tml_tm_count: 1, tml_cache_count: 1, tml_sql_count: 1, tml_log_count: 1, tml_nosql_count: 1,
Small	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 2	tml_cm_count: 1, tml_tm_count: 3, (max 2 per node) tml_cache_count: 2, (1 per node) tml_sql_count: 1, tml_log_count: 1, tml_nosql_count: 1,

Topology	Kubernetes Configuration	Pod Configuration
	k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	
Small-2	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 2, (1 node dedicated to 1 TM) k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	tml_cm_count: 1, tml_tm_count: 1, tml_cache_count: 1, tml_sql_count: 1, tml_log_count: 1, tml_nosql_count: 1,
Medium-1	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 3, k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	tml_cm_count: 1, tml_tm_count: 10, (max 4 per node) tml_cache_count: 3, (1 per node) tml_sql_count: 1, tml_log_count: 2, (max 1 per node) tml_nosql_count: 3, (1 per node)
Medium-2	medium-2 cluster of 3 nodes with 10 TMs (same	

Topology	Kubernetes Configuration	Pod Configuration
	as medium-1). Here, nodes are 8 core and 30 GB.	
Medium-3	medium-2 cluster of 3 nodes with 3 TMs (same as medium-1). Here, each node is 2 core and 30 GB. This test has been done to get number for licensing for total of 6 core.	tml_cm_count: 1, tml_tm_count: 3, (max 1 per node) tml_cache_count: 2, (1 per node) tml_sql_count: 1, tml_log_count: 2, (max 1 per node) tml_nosql_count: 3, (1 per node)
Large-1	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 5, k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	tml_cm_count: 1, tml_tm_count: 20, (max 4 per node) tml_cache_count: 3, (max 1 per node) tml_sql_count: 1, tml_log_count: 5, (max 1 per node) tml_nosql_count: 3, (max 1 per node)
Large-2	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 6 (3 Dedicated for TM),	tml_cm_count: 1, tml_tm_count: 15, (max 5 per node) tml_cache_count: 3, (max 1 per node) tml_sql_count: 1, tml_log_count: 2, (max 1 per node) tml_nosql_count: 3, (max 1 per node)

Topology	Kubernetes Configuration	Pod Configuration
	k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	
Xtra Large-1	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 8, k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	tml_cm_count: 1, tml_tm_count: 40, (max 5 per node) tml_cache_count: 5, (max 1 per node) tml_sql_count: 1, tml_log_count: 3, (max 1 per node) tml_nosql_count: 5, (max 1 per node)
Xtra Large-2	k8s_master_count: 1, k8s_master_size: "t2.medium", k8s_master_volume_size: 32, k8s_node_count: 10 (5 dedicated for TM), k8s_node_size: "t2.xlarge", k8s_node_volume_size: 32,	tml_cm_count: 1, tml_tm_count: 40, (max 8 per node) tml_cache_count: 5, (max 1 per node) tml_sql_count: 1, tml_log_count: 3, (max 1 per node) tml_nosql_count: 5, (max 1 per node)

Switching from Tethered to Untethered Mode

The following section describes migrating from tethered mode to untethered mode.

In tethered mode, configuration of API is done in Cloud API Management (CAM) Control Center. The configuration data is stored in Cloud API Management database and synced to on-prem clusters via Mashery Onprem Manager (MOM) periodically.

In untethered mode, configuration of API is done in the Local Edition Configuration UI and configuration data is stored in the local database.

If all data in the Local Edition cluster needs to be in the local cluster, the process of migrating to untethered will download existing data from tethered mode to local cluster.

Switching from tethered to untethered mode is supported when:

- Migrating from version 5.3.x, 5.4.x, 5.5.x, 5.6.0 clusters in tethered mode to 5.6.x untethered mode.

Limitations

Switching from tethered to untethered is a one-way conversion. Once the cluster is switched to untethered, switching back to tethered is not supported. This is because after switching to untethered mode:

- The area ID of configurations from Cloud APIM (MOM) are reset to Local Edition default area id (2222)
- Any new or update to configuration of Local Edition Control Center are not propagate back to Cloud APIM.

During migration, configuration changes should not be done in Local Edition Config UI as changes made in Local Edition Config UI may be overwritten from the changes in Cloud APIM Control Center.

Also, the audit log history is not migrated to the Local EditionCluster. As a result, the history for changes (for example, to services and packages) are not available after migration.

Important: After moving to untethered, any Cloud APIM reporting features will no longer be available.

Migration Flow

When preparing for migration, keep in mind that the migrated cluster must have all the configuration data needed from existing SQL database in the tethered mode cluster prior to the migration. The migrated cluster must have all the OAuth tokens from existing NoSQL database in the tethered mode cluster prior to the migration. Then, the migrated untethered cluster must have all the API configuration data and OAuth tokens from Cloud APIM.

Overall, the migration is done similar to rolling upgrade for all components with the exception of SQL database, due to the SQL database having the replication group enabled in the migrated cluster.

1. Prepare the existing tethered mode Local Edition 5.X cluster for migration.
    a. Backup HTTPs client profiles from SQL. See the details in "Dump MySQL Data from the Old TML Cluster" of Upgrade to TML 5.5.2 K8S from TML 5.6.0 K8S cluster (Tethered). The backup will be restored to new cluster in a subsequent step.
    b. Note the existing NoSQL persistence volume configuration in new cluster. The persistence volume is reused in the new cluster.
2. Bring up the new Local Edition 5.6 cluster in untethered mode.
    a. Specify the MOM key and MOM secret for the tethered area in the cluster configuration.
    b. Reuse the existing NoSQL persistence volume configuration in new cluster from step 1.
3. Restore the HTTPs client profiles to SQL in the new cluster. See the details in "Copy MySQL Data dumped from the Old TML Cluster to the New TML Cluster" and, "Import MySQL Data dumped from the Old TML Cluster to the new TML Cluster" of Upgrade to TML 5.5.2 K8S from TML 5.6.0 K8S cluster (Tethered).
4. Verify that all containers in new cluster are ACTIVE. To verify cluster status, run the following command in the CM container/pod:

```
clustermanager list components
```

5. Initiate a configuration data download with onpremloader API. To initiate a full download of configuration data from MOM, use the following curl call:

```
curl -v -XPUT "http://127.0.0.1:8080/onpremloader?apiKey={momkey}&apiSecret=
{momSecret}&switchTo=tethered"
```

   For example:

```
curl -v -XPUT
"http://127.0.0.1:8080/onpremloader?downloadAndSync=true&apiKey=REDACTED&a
piSecret=REDACTED&switchTo=tethered"
```

apiKey and apiSecret are required to enable the MOM sync if they are not specified in the step 2 above.

6. Initiate a full cache load populate the cache. A request to onpremloader is needed to initiate loading of data from MOM, which is needed to be done in the Cache container or pod. To initiate a full cache load, use the following call.

```
/opt/javaproxy/proxy/cacheloader --service --mapi --devclass --packager --
httpsclientsecurity --env production --verbose
```

7. Verify that configuration is migrated. Login to APIM Local Control Center UI to verify that API Definition, Packages, Applications, etc are available.
8. Verify that protected calls are working with new and existing tokens. Create a new token and make calls to a migrated protected endpoint and verify that request is successful.
9. Switch back to untethered mode. To switch to untethered mode after configuration data are download and verify, run:

```
curl -v -XPUT "http://127.0.0.1:8080/onpremloader?switchTo=untethered""
```

10..Configure new services in Local Edition Configuration Manager UI.

11..Verify that new services are working.

Migration for all versions of 5.3.X, 5.4.X, 5.5.X (including 5.6.0) tethered to 5.6 untethered follow the same steps.

After migration, data in Cloud APIM will remain in Cloud APIM and will not be synced to Local Edition after switching to untethered mode.

Configuring SSL for Node to Node Connections

The following section describes how to configure SSL for node-to-node connections.

New Properties

The following new properties have been added to **tml_nosql_properties.json** to control the *server_encryption_options*:

```
"server_internode_encryption": "none",
```

```
"server_keystore_password": "changeme",
```

```
"server_truststore_password": "changeme",
```

```
"server_require_client_auth": false
```

For the *server_encryption_options* section of the **cassandra.yaml** file, the following options are supported for the *server_internode_encryption* setting:

- none - No encryption (default)
- all - Encrypt all internode communications

For the *server_require_client_auth* setting:

- When set to "false", client auth in Cassandra internode communication is not required.
- When set to "true", client auth in Cassandra internode communication is required.

Added Reference Keystore and Truststore

In the reference deployment scripts, under folder "properties":

- **tml-nosql.p12**: sample keystore for tml-nosql

- **tml-nosql-trust.p12**: sample truststore for tml-nosql

How to Prepare Keystore and Truststore

Note:

- There are other ways to prepare the keystore and truststore, for example, using OpenSSL.
- For additional reference, see "Creating local SSL certificate and keystore files."

Root CA

Create a New keystore for Root CA

File => New to created a PKCS #12 format keystore

Generate Root CA

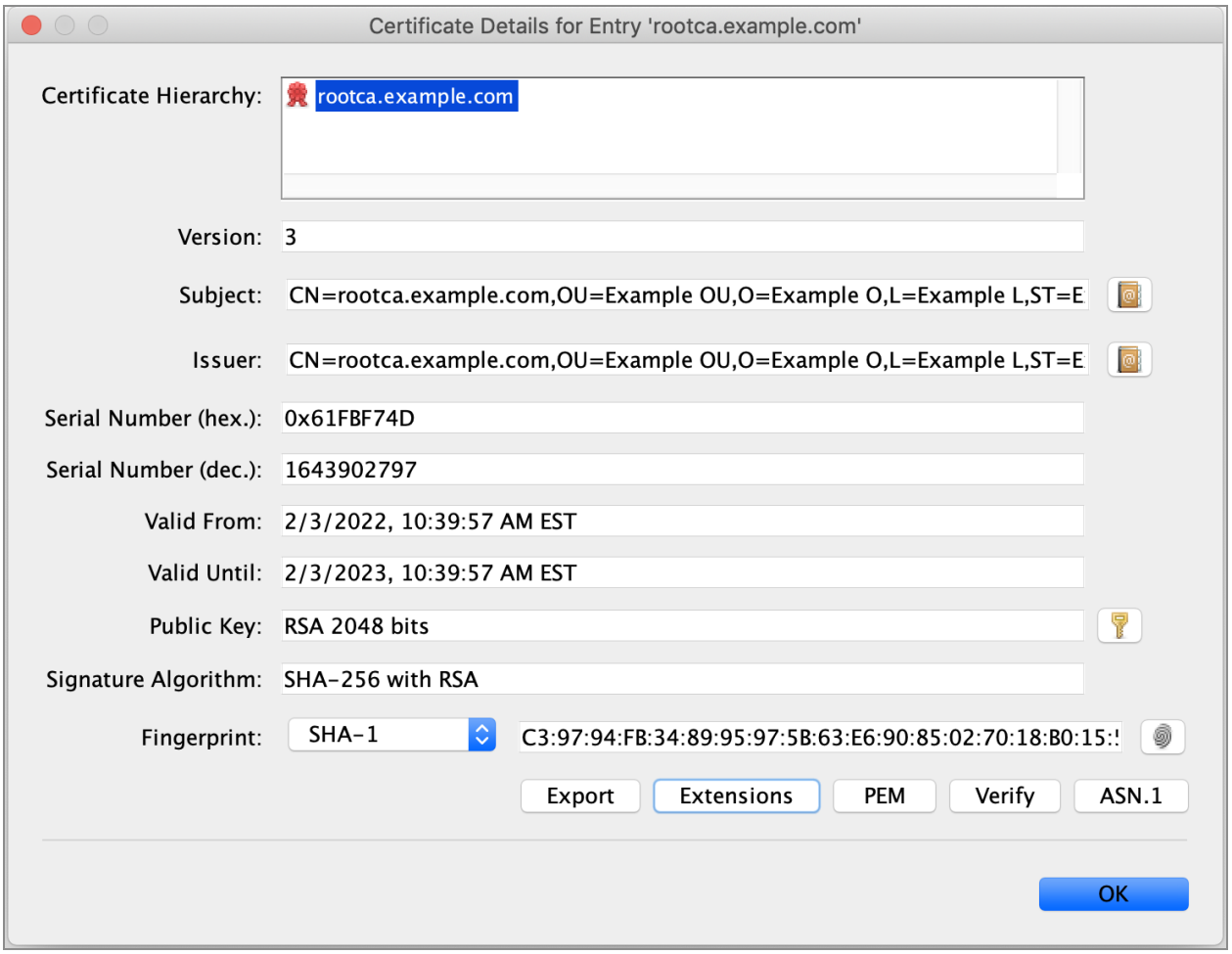
Tools => Generate Key Pair to generate a new key pair as root CA

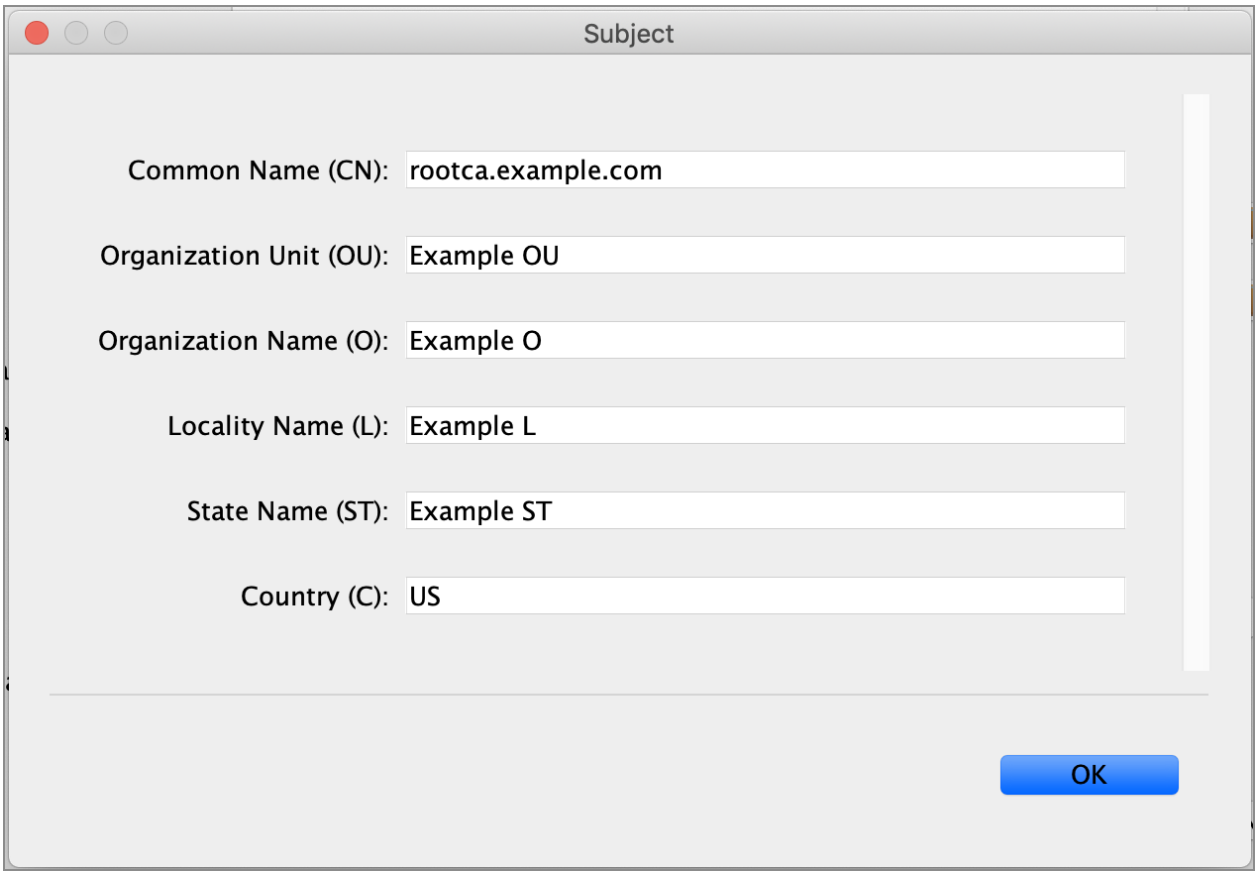
Algorithm: RSA

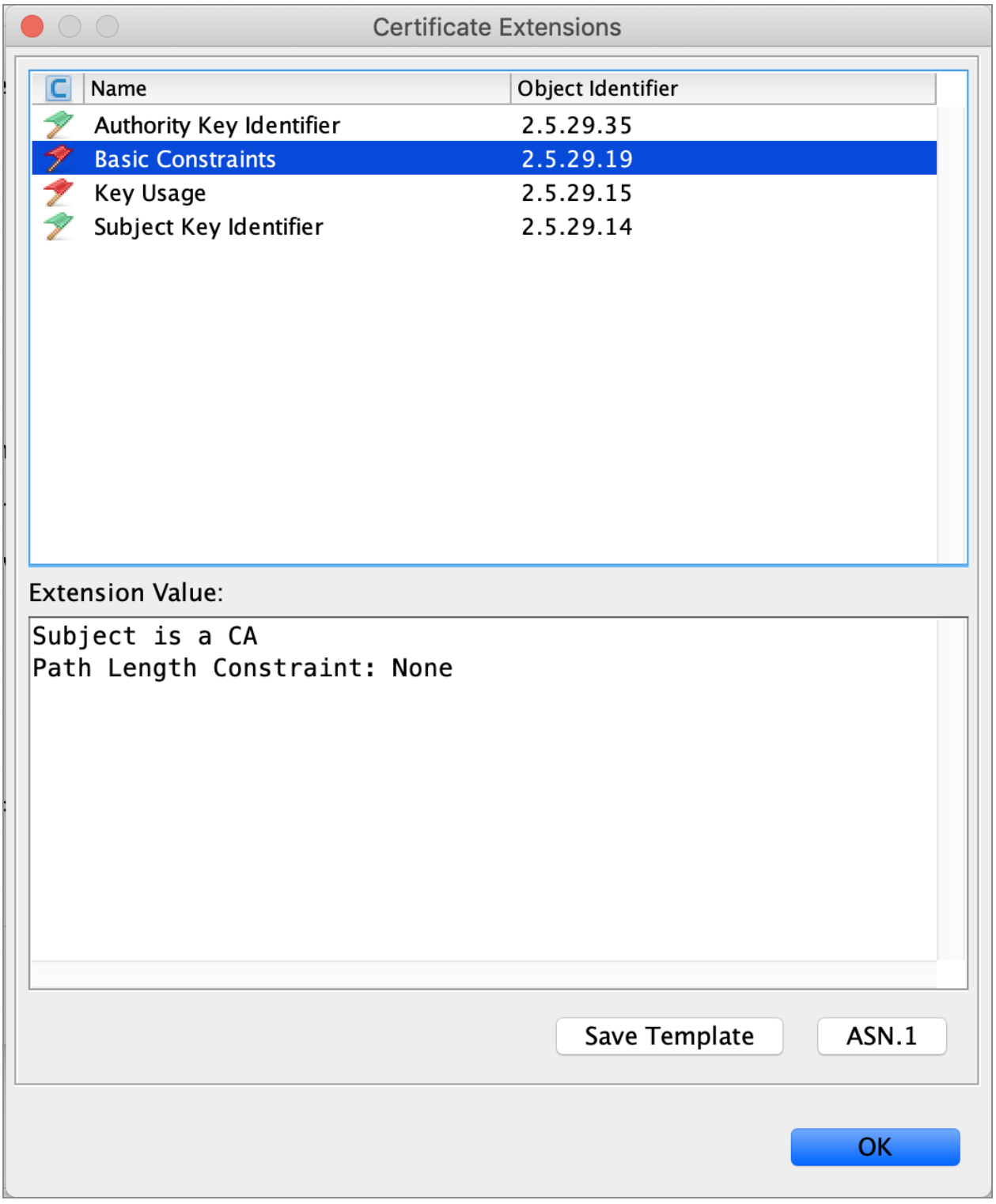
Key Size: 2048

Validity Period: 10 years

Extension: Use Standard Template, select "CA"

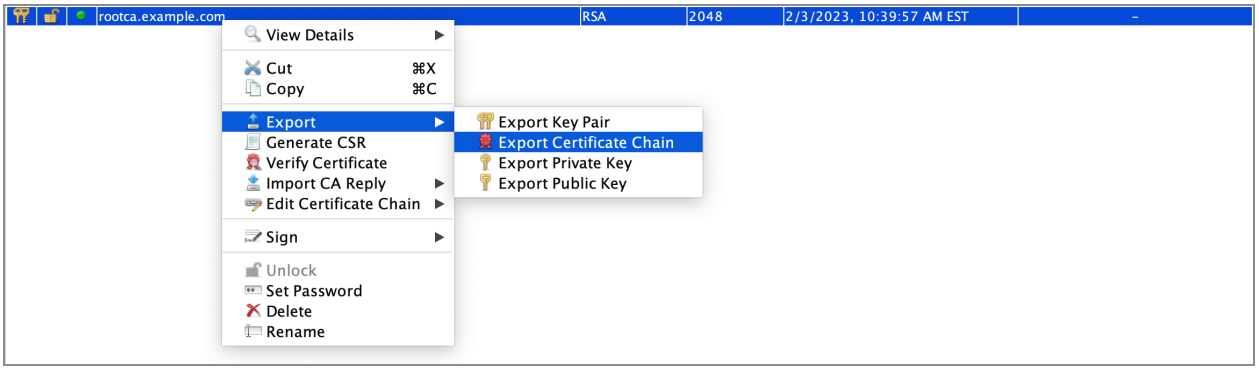

Certificate Details for Entry 'rootca.example.com'
Certificate Hierarchy:
rootca.example.com
Version:
3
Subject:
CN=rootca.example.com,OU=Example OU,O=Example O,L=Example L,ST=E
Issuer:
CN=rootca.example.com,OU=Example OU,O=Example O,L=Example L,ST=E
Serial Number (hex.):
0x61FBF74D
Serial Number (dec.):
1643902797
Valid From:
2/3/2022, 10:39:57 AM EST
Valid Until:
2/3/2023, 10:39:57 AM EST
Public Key:
RSA 2048 bits
Signature Algorithm:
SHA-256 with RSA
Fingerprint:
SHA-1
C3:97:94:FB:34:89:95:97:5B:63:E6:90:85:02:70:18:B0:15:
Export
Extensions
PEM
Verify
ASN.1
OK


Subject
Common Name (CN):
rootca.example.com
Organization Unit (OU):
Example OU
Organization Name (O):
Example O
Locality Name (L):
Example L
State Name (ST):
Example ST
Country (C):
US
OK

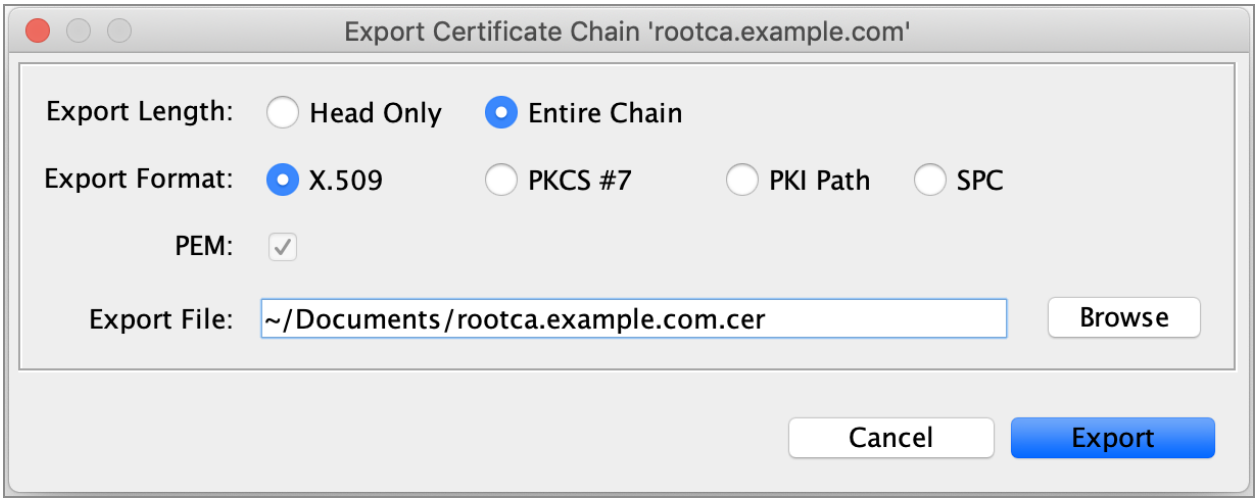

Certificate Extensions
Name
Object Identifier
Authority Key Identifier
2.5.29.35
Basic Constraints
2.5.29.19
Key Usage
2.5.29.15
Subject Key Identifier
2.5.29.14
Extension Value:
Subject is a CA
Path Length Constraint: None
Save Template
ASN.1
OK

Truststore

Export Root CA Certificate Chain from Root CA Keystore



Here is the dialog for exporting certificate chain:



Create a New Keystore in PKCS #12 format to be used as Truststore

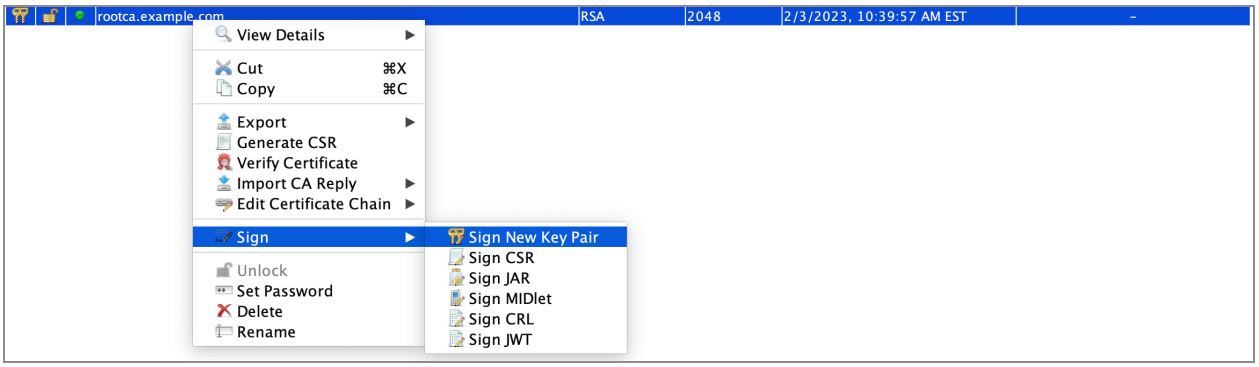
File => New

Import the Exported Root CA Certificate Chain

Tools => Imported Trusted Certificate

Keystore

Use Root CA to Sign a New Key Pair



Algorithm: RSA

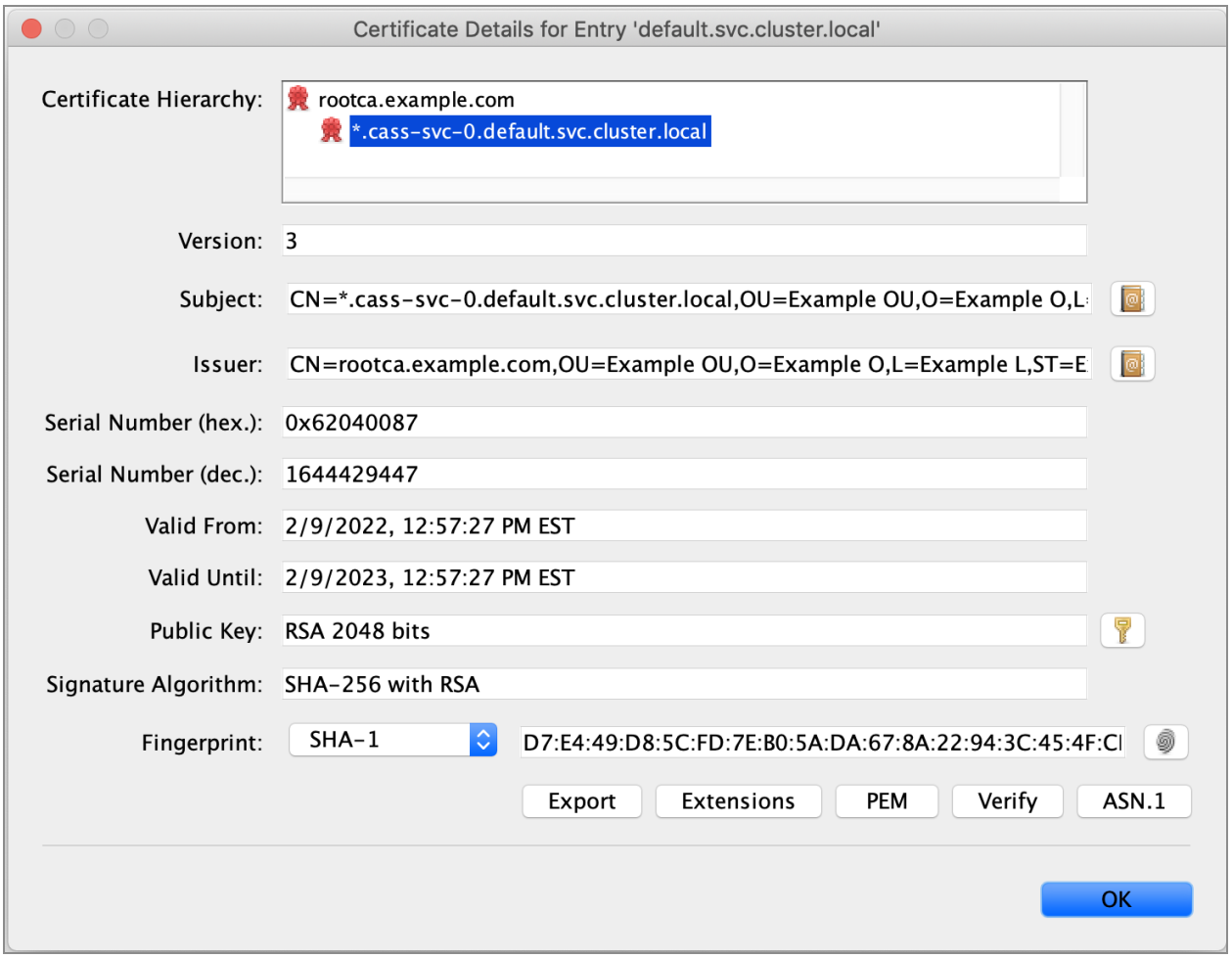
Key Size: 2048

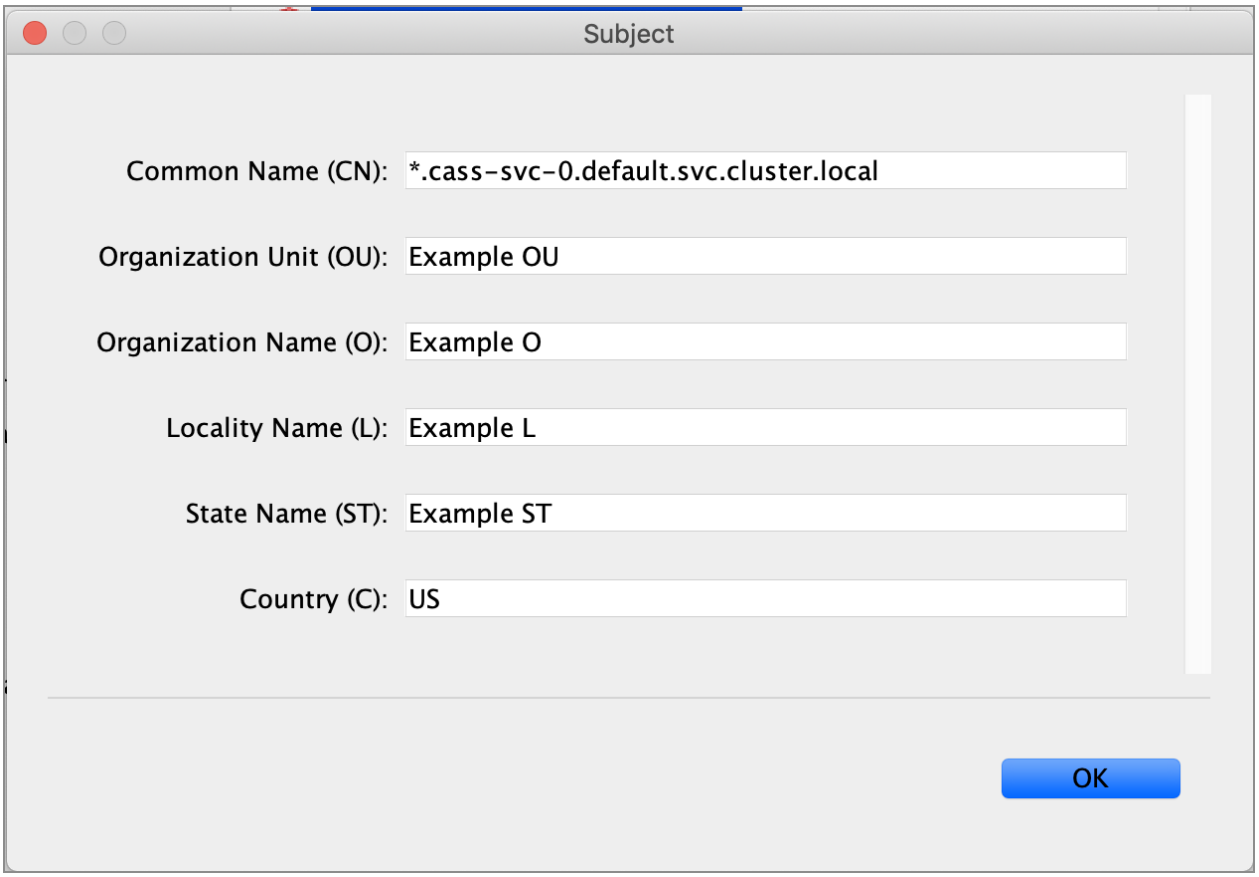
Validity: 10 years

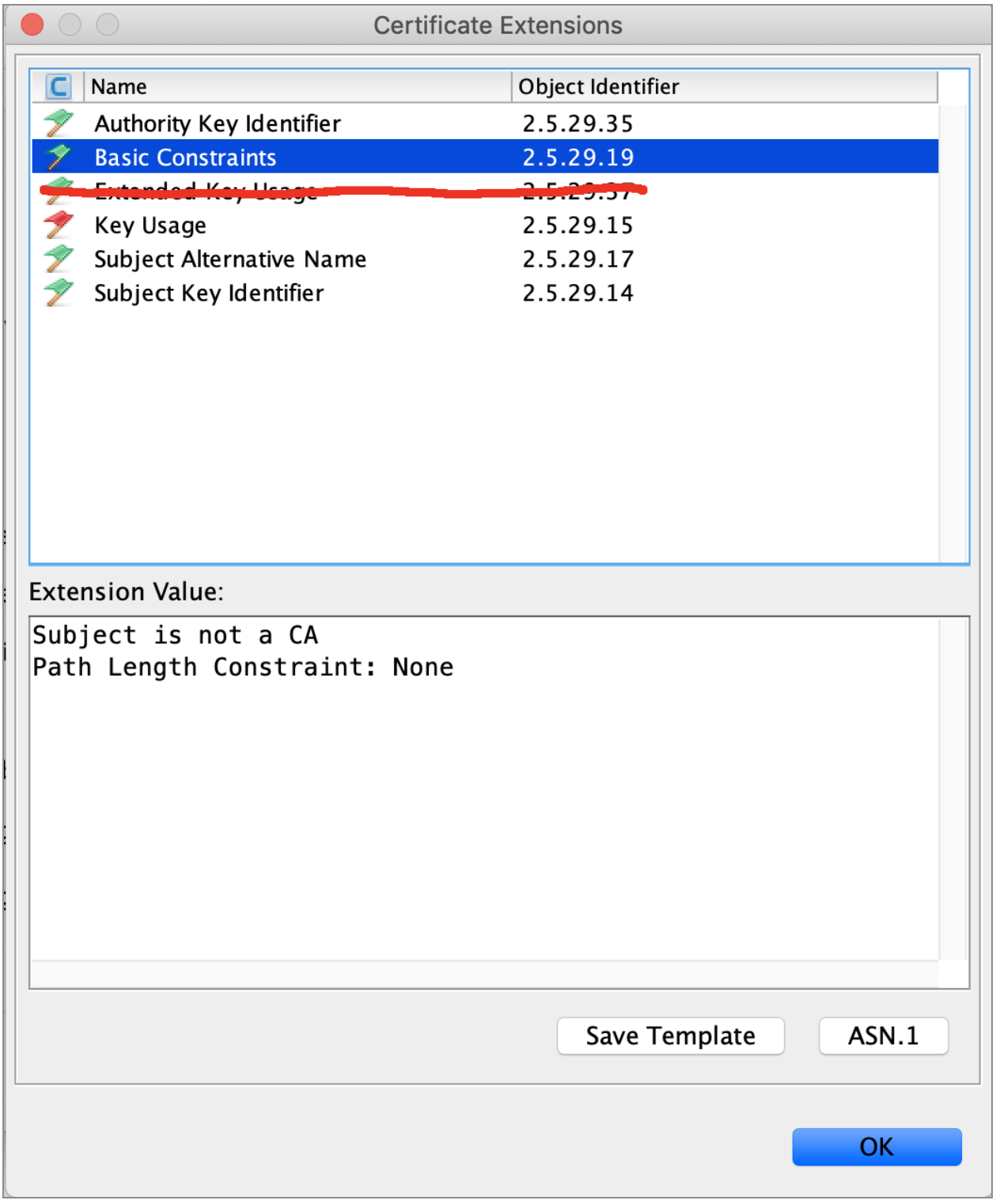
Extensions:

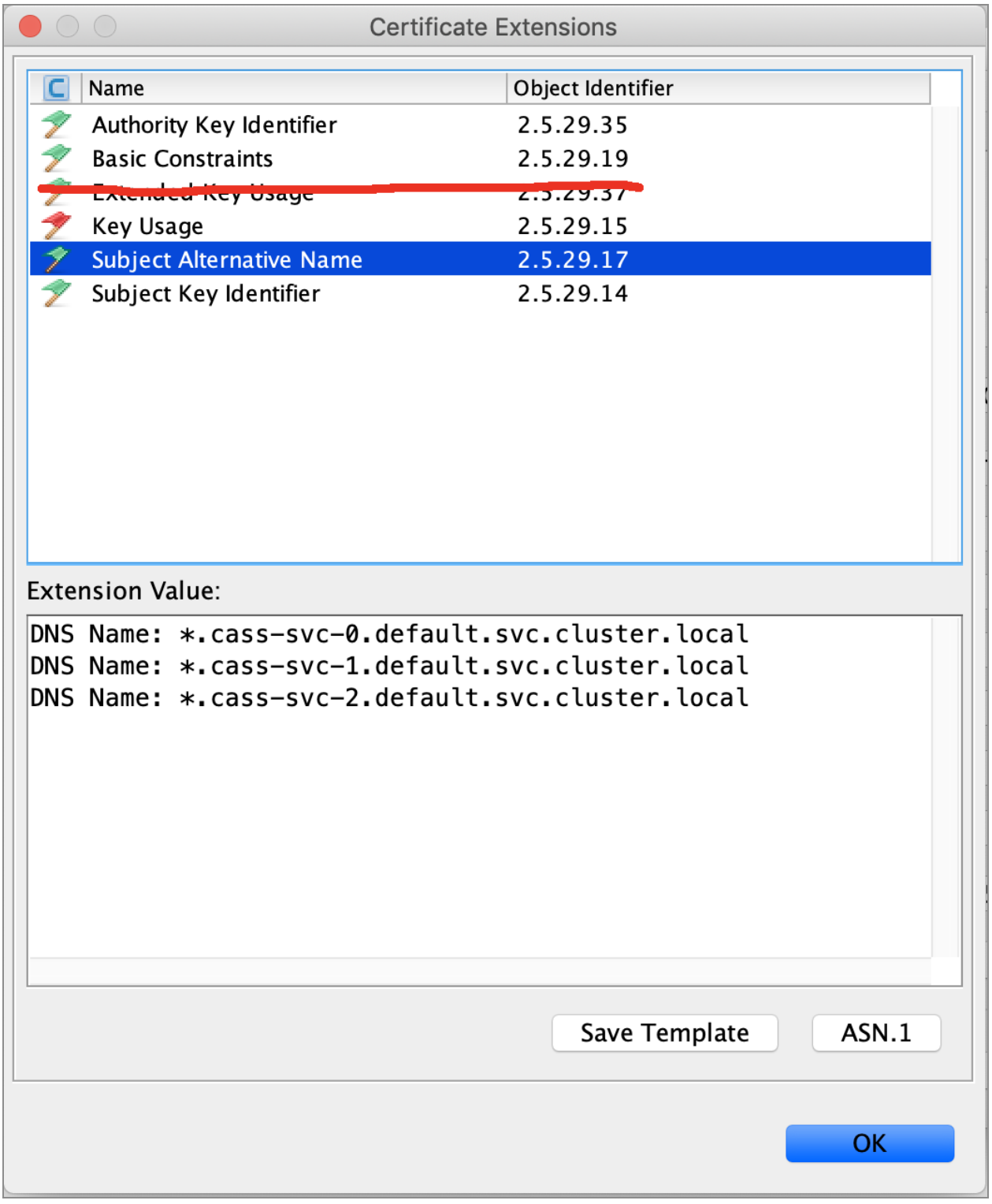
- use Standard Template "SSL Client";
- remove "Extended Key Usage";
- add "Basic Constraints" extension;
- add "Subject Alternative Name" extension with the following contents:
  - DNS Name: *.cass-svc-0.default.svc.cluster.local
  - DNS Name: *.cass-svc-1.default.svc.cluster.local
  - DNS Name: *.cass-svc-2.default.svc.cluster.local

Save the key pair with entry name "default.svc.cluster.local"


Certificate Details for Entry 'default.svc.cluster.local'
Certificate Hierarchy:
rootca.example.com
*.cass-svc-0.default.svc.cluster.local
Version:
3
Subject:
CN=*.cass-svc-0.default.svc.cluster.local,OU=Example OU,O=Example O,L
Issuer:
CN=rootca.example.com,OU=Example OU,O=Example O,L=Example L,ST=E
Serial Number (hex.):
0x62040087
Serial Number (dec.):
1644429447
Valid From:
2/9/2022, 12:57:27 PM EST
Valid Until:
2/9/2023, 12:57:27 PM EST
Public Key:
RSA 2048 bits
Signature Algorithm:
SHA-256 with RSA
Fingerprint:
SHA-1
D7:E4:49:D8:5C:FD:7E:B0:5A:DA:67:8A:22:94:3C:45:4F:CI
Export
Extensions
PEM
Verify
ASN.1
OK


Subject
Common Name (CN):
*.cass-svc-0.default.svc.cluster.local
Organization Unit (OU):
Example OU
Organization Name (O):
Example O
Locality Name (L):
Example L
State Name (ST):
Example ST
Country (C):
US
OK

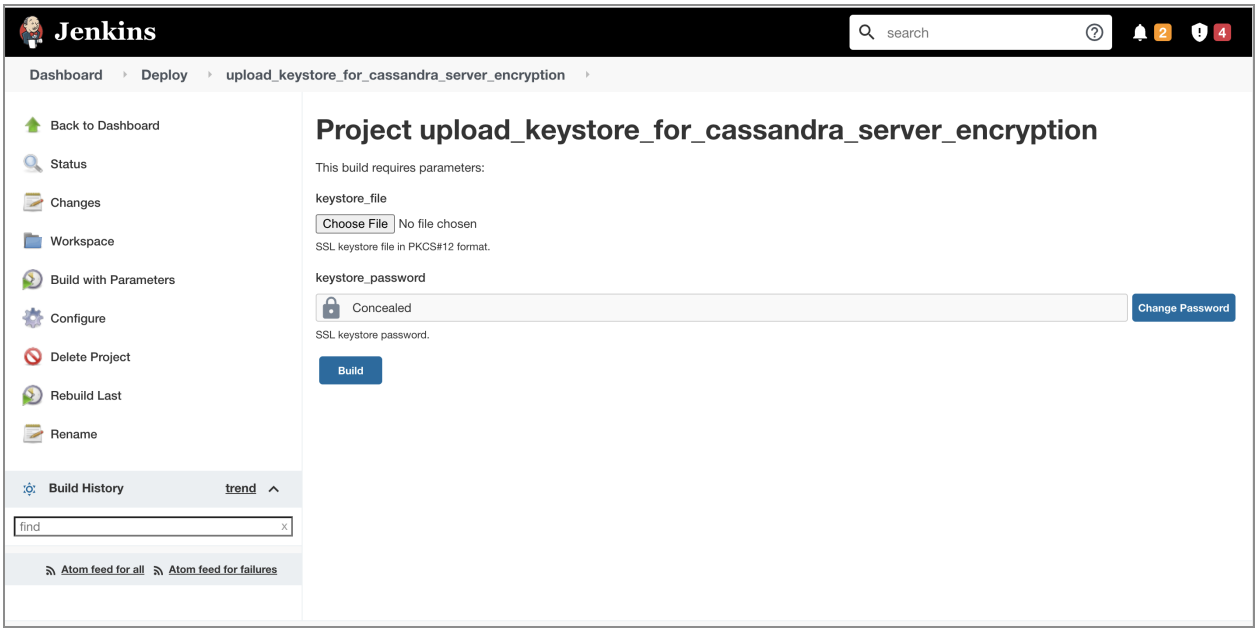

Certificate Extensions
Name
Object Identifier
Authority Key Identifier
2.5.29.35
Basic Constraints
2.5.29.19
Key Usage
2.5.29.15
Subject Alternative Name
2.5.29.17
Subject Key Identifier
2.5.29.14
Extension Value:
Subject is not a CA
Path Length Constraint: None
Save Template
ASN.1
OK


Certificate Extensions
Name
Object Identifier
Authority Key Identifier
2.5.29.35
Basic Constraints
2.5.29.19
Key Usage
2.5.29.15
Subject Alternative Name
2.5.29.17
Subject Key Identifier
2.5.29.14
Extension Value:
DNS Name: *.cass-svc-0.default.svc.cluster.local
DNS Name: *.cass-svc-1.default.svc.cluster.local
DNS Name: *.cass-svc-2.default.svc.cluster.local
Save Template
ASN.1
OK

Upload Keystore and Truststore in Local Edition Installer

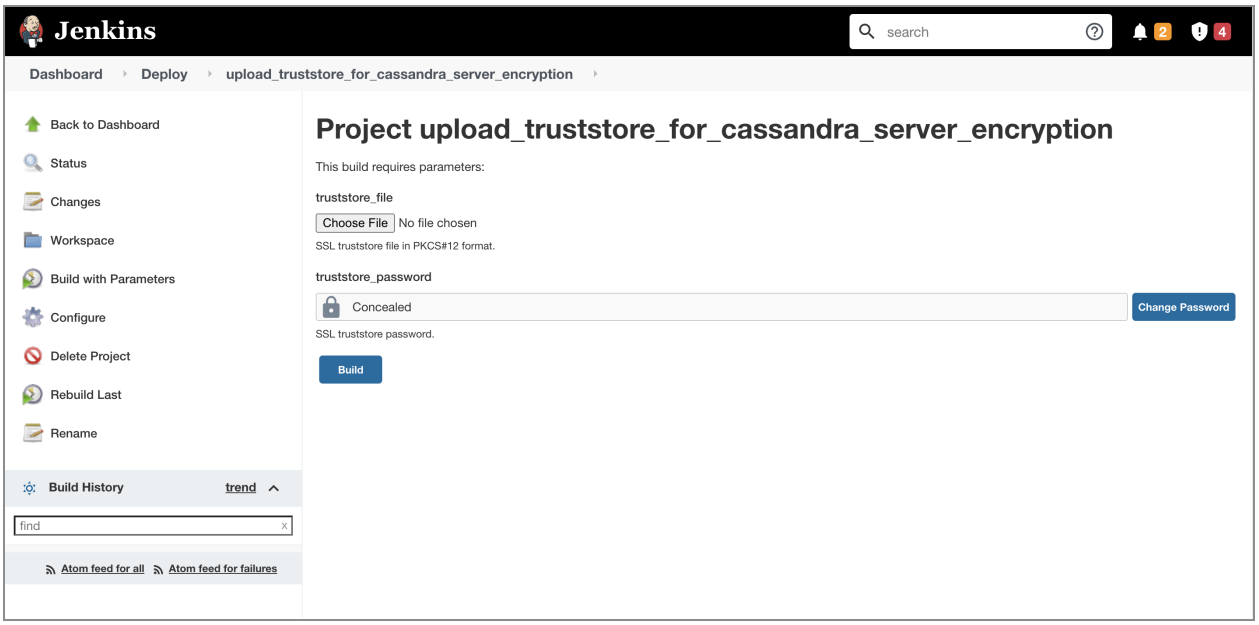
Upload Keystore in Local Edition Installer

Use the following Jenkins job in Local Edition Installer to upload keystore for Cassandra server encryption.



Upload Truststore in Local Edition Installer

Use the following Jenkins job in Local Edition Installer to upload truststore for Cassandra server encryption.


Jenkins
search
2
4
Dashboard
Deploy
upload_truststore_for_cassandra_server_encryption
Back to Dashboard
Status
Changes
Workspace
Build with Parameters
Configure
Delete Project
Rebuild Last
Rename
Build History
trend
find
Atom feed for all
Atom feed for failures
Project upload_truststore_for_cassandra_server_encryption
This build requires parameters:
truststore_file
Choose File
No file chosen
SSL truststore file in PKCS#12 format.
truststore_password
Concealed
Change Password
SSL truststore password.
Build

Boomi References

Refer to these links to learn more about Boomi privacy policy, terms of service, and Boomi help documentation:

Privacy Policy
Terms of Service
Help Documentation