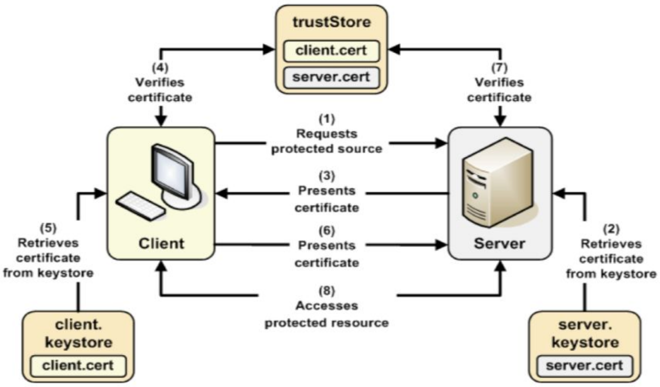
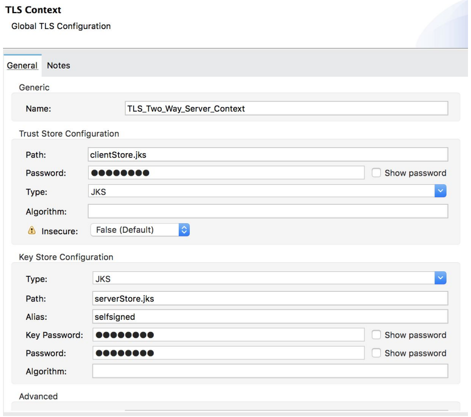
Module 16: Securing Network Communications between Mule Applications

Goal





TLS Context

Global TLS Configuration

Generic

Name: TLS_Two_Way_Server_Context

Trust Store Configuration

Path: clientStore.jks

Password: ●●●●●●●● □ Show password

Type: JKS

Algorithm:

Insecure: False (Default)

Key Store Configuration

Type: JKS

Path: serverStore.jks

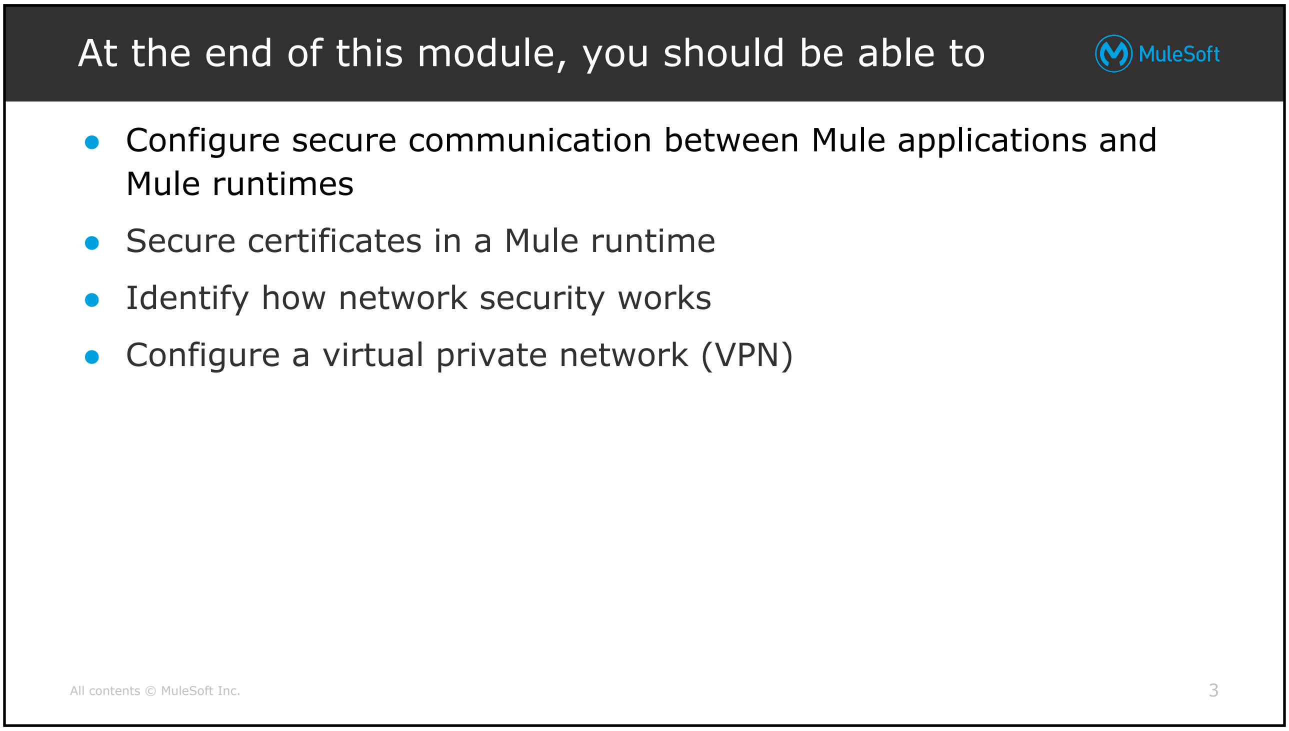
Alias: selfsigned

Key Password: ●●●●●●●● □ Show password

Password: ●●●●●●●● □ Show password

Algorithm:

Advanced



At the end of this module, you should be able to

- Configure secure communication between Mule applications and Mule runtimes
- Secure certificates in a Mule runtime
- Identify how network security works
- Configure a virtual private network (VPN)

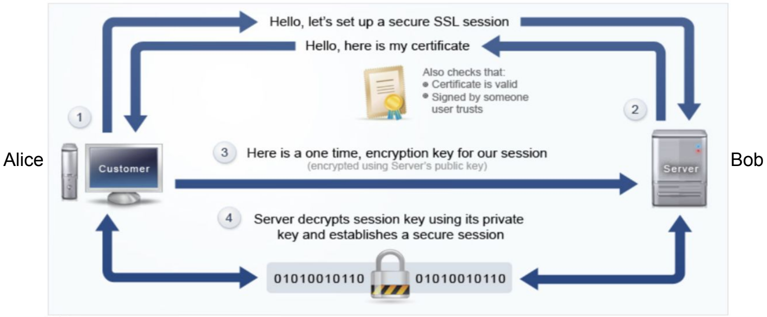
Securing Mule application communications

How transport layer security (TLS) is used with Mule applications

- TLS **encrypts** data sent to and from Mule applications and other systems or applications
- **Certificates** are used for one-way or two-way handshakes
  - Certificates are used for **asymmetric public/private key** cryptography and to verify and trust the identity of a client, server, or other entity
  - In a one-way handshake the client must trust the server's identity
  - In a two-way handshake, the server must also trust the client's identity
  - Trust is often established by a chain of trusted certificate signing authorities
- Data **encrypted** with the public key can **only** be **decrypted** with the **private** key

Reviewing asymmetric cryptography

- Asymmetric cryptography uses a public/private key pair
  - A message encrypted with the public key can only be decrypted with the corresponding private key
  - A message encrypted (signed) by a private key can only be decrypted with the corresponding public key
  - The public key is usually exchanged with one or more other parties
- This is used to secure communications, such as with SSL or TLS

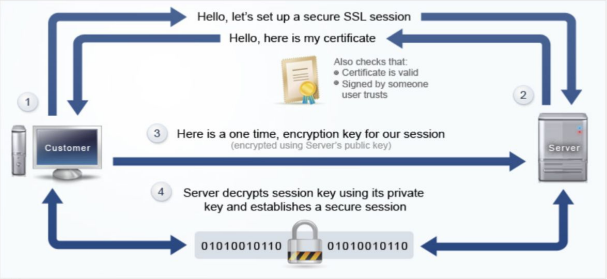

Hello, let's set up a secure SSL session
Hello, here is my certificate
Also checks that:
● Certificate is valid
● Signed by someone user trusts
1
2
Alice
Customer
3 Here is a one time, encryption key for our session
(encrypted using Server's public key)
Server
Bob
4 Server decrypts session key using its private key and establishes a secure session
01010010110
01010010110

What is contained in a public certificate

- A **public certificate** contains credentials
  - The public key (but not the private key)
  - Organization details
  - The certificate issuer
- A certificate is typically signed by a trusted Certificate Authority (**CA**) or can be self signed using the Java keytool
  - Other people or systems can use the trusted CA's public key to validate the authenticity of the CA signature, and hence the certificate

How asymmetric cryptography is used to authenticate (trust) publicly available data

2. The server has a **certificate** (public key) that is signed by a trusted certificate authority (CA) using the CA's private key
   - The server sends its **signed** (encrypted) **certificate** to the client
   - The client uses the CA's public key to decrypt/**authenticate** the server's certificate
   - This guarantees the server certificate (public key) is **valid** and **untampered**
   - The client now **trusts** the server is the only entity with the private key corresponding to the **server** certificate's **public key**



Hello, let's set up a secure SSL session

Hello, here is my certificate

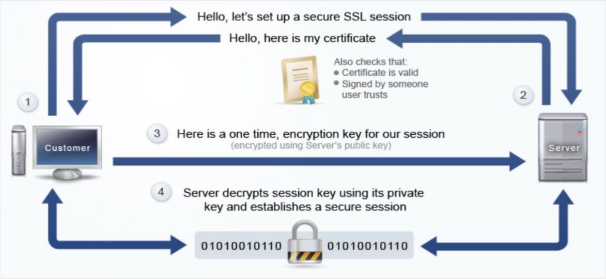
Also checks that:
- Certificate is valid
- Signed by someone user trusts

3 Here is a one time, encryption key for our session (encrypted using Server's public key)

4 Server decrypts session key using its private key and establishes a secure session

How asymmetric cryptography is used to initiate a secure communication channel

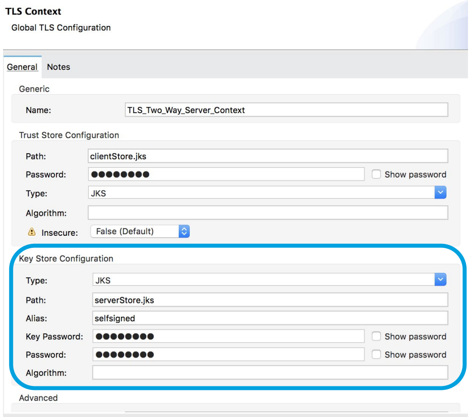
3. The client **encrypts** a **symmetric key** into a message using the server **certificate** (public key), then **safely** sends the message to the server
4. Only the server has the **private key** to unlock the symmetric key in the message
   - No one "in the middle" can steal the symmetric key
   - Even the client cannot decrypt the message, but the client still has the original version
   - Now the client and server can safely communicate over the internet using the same symmetric key to encrypt and decrypt messages, which is faster


Hello, let's set up a secure SSL session
Hello, here is my certificate
Also checks that:
• Certificate is valid
• Signed by someone user trusts
1
2
Customer
Server
3 Here is a one time, encryption key for our session
(encrypted using Server's public key)
4 Server decrypts session key using its private key and establishes a secure session
01010010110
01010010110

Securing Mule applications using Java key stores

Understanding Java keystores

- A **keystore** stores public certificates plus corresponding private keys (credential) for clients or servers in a Mule application
- The Java keytool can be used to create keystores for Mule applications or Anypoint Platform
- Public certificates in a keystore have a **certificate chain associated** with them, which **authenticates** the corresponding public key
- In TLS, the keystore determines the credentials (public certificate) sent to the remote host (e.g., client)



TLS Context
Global TLS Configuration

General Notes

Generic
Name: TLS_Two_Way_Server_Context

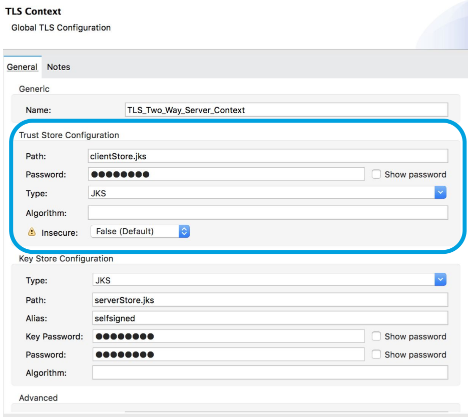
Trust Store Configuration
Path: clientStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration
Type: JKS
Path: serverStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced

How truststores are created and used Anypoint Platform and Mule applications

- The **Java keytool** is used to create **truststores**
- The **truststore** contains **public certificates** (self signed or from from a CA) for remote hosts (other parties) and perhaps also the signing CAs



TLS Context
Global TLS Configuration

General Notes

Generic
Name: TLS_Two_Way_Server_Context

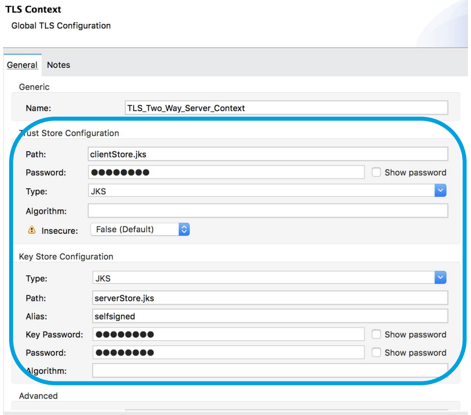
Trust Store Configuration
Path: clientStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration
Type: JKS
Path: serverStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced

Understanding Java truststores

- The **keystore owner** trusts the **public certificates** (and the contained public keys) contained in its truststore to identify other parties
  - Each other party must use the correct corresponding private key
  - Each private key is **not** stored in the truststore and is usually **not** available to the truststore owner



TLS Context

Global TLS Configuration

Generic

Name: TLS_Two_Way_Server_Context

Trust Store Configuration

Path: clientStore.jks

Password: ●●●●●●●● □ Show password

Type: JKS

Algorithm:

⚠ Insecure: False (Default)

Key Store Configuration

Type: JKS

Path: serverStore.jks

Alias: selfsigned

Key Password: ●●●●●●●● □ Show password

Password: ●●●●●●●● □ Show password

Algorithm:

Advanced

- In TLS, the truststore determines whether credentials (public certificates) sent by the remote host (the client) are trusted and hence if the secure connection can be established

How to configure the **insecure** parameter in a truststore

- The **insecure** parameter in a truststore element determines whether or not to validate the trust-store
- If set to true, no validation occurs
  - By default **insecure = false**
  - Setting **insecure = true** renders connections **vulnerable to attacks** and is recommended only for prototyping and testing purposes
    - For example, to troubleshoot security related issues, where the error and log messages are necessarily cryptic, turning off security features then incrementally adding them back can be helpful

Implementing TLS communications in Mule applications

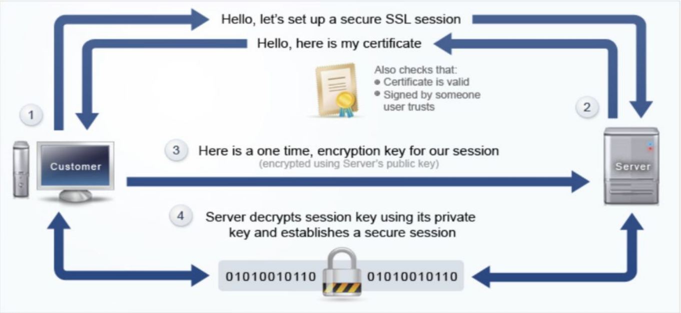
Anypoint connectors that support TLS connections

MuleSoft

- HTTP/S client and server
  - one-way or two-way TLS communication
- SFTP client
- SMTP/S clients
- TCP Sockets client and server
- JMS client
  - The JMS standard does not specify TLS connections
  - The JMS provider must support TLS and provide a TLS connection factory in its client library

Support for TLS in HTTP/S connector

- One-way TLS
  - The server sends its identity (public certificate containing its public key) to the client
  - The client uses the server identity to safely exchange a symmetric key for private and encrypted two-way communication
    - Only the server can decrypt this key (man-in-the-middle attacks will fail)
  - The server is not concerned (and ignores) the client's identity



1 Hello, let's set up a secure SSL session

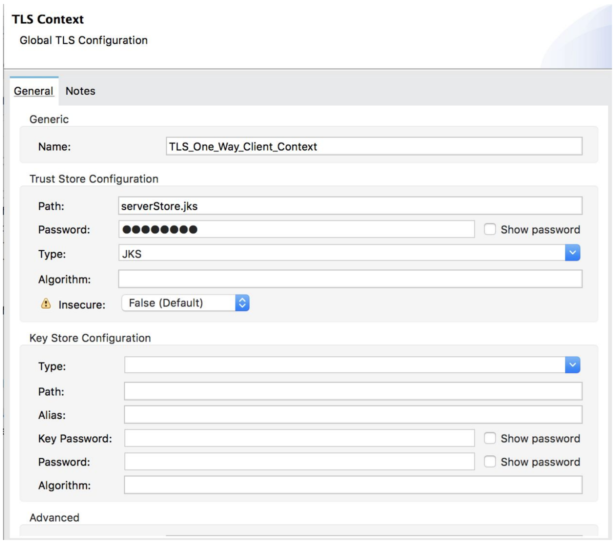
2 Hello, here is my certificate

Also checks that:
- Certificate is valid
- Signed by someone user trusts

3 Here is a one time, encryption key for our session (encrypted using Server's public key)

4 Server decrypts session key using its private key and establishes a secure session

Configuring one-way TLS for HTTPS



TLS Context
Global TLS Configuration

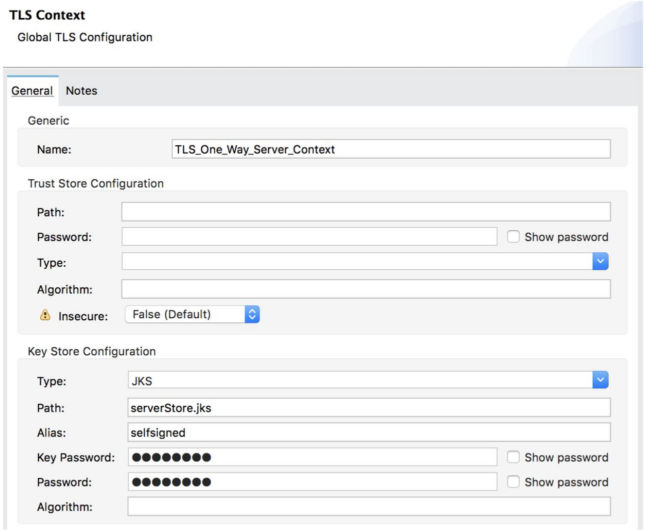
General | Notes

Generic
Name: TLS_One_Way_Client_Context

Trust Store Configuration
Path: serverStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration
Type:
Path:
Alias:
Key Password: □ Show password
Password: □ Show password
Algorithm:

Advanced



TLS Context
Global TLS Configuration

General | Notes

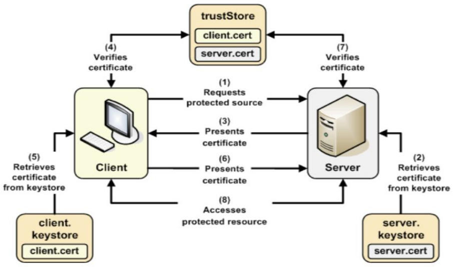
Generic
Name: TLS_One_Way_Server_Context

Trust Store Configuration
Path:
Password: □ Show password
Type:
Algorithm:
Insecure: False (Default)

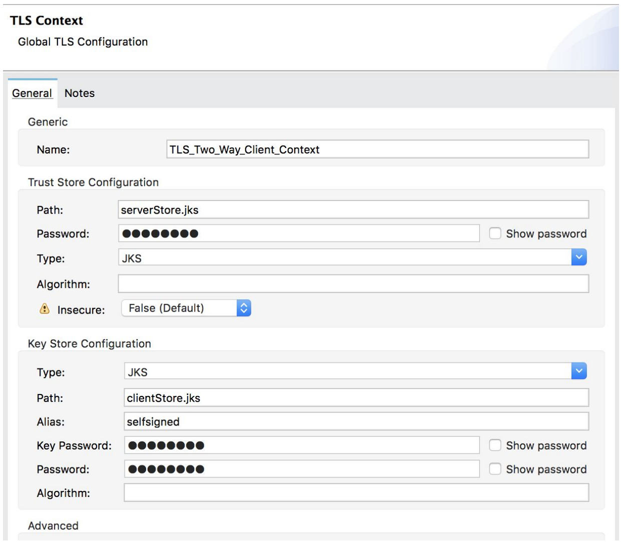
Key Store Configuration
Type: JKS
Path: serverStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Support for two-way TLS in the HTTP/S connector

- Two-way TLS
  - In this model, the client also sends its identity (public certificate, containing public key) to the server
  - The client has its own keystore (with a private key)
  - Optionally, may also include a truststore containing the server's public certificate
    - This is not required if the server certificate is signed by a well known CA
  - Server has its own truststore to validate the client's certificate



Configuring two-way TLS for HTTPS



TLS Context
Global TLS Configuration

General | Notes

Generic

Name: TLS_Two_Way_Client_Context

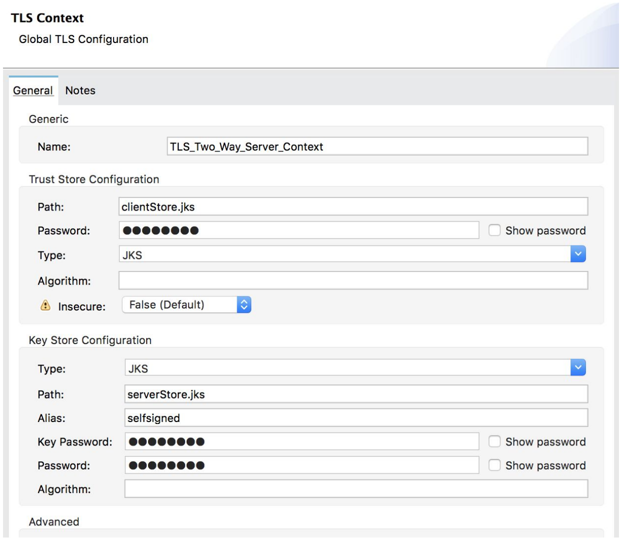
Trust Store Configuration

Path: serverStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration

Type: JKS
Path: clientStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced



TLS Context
Global TLS Configuration

General | Notes

Generic

Name: TLS_Two_Way_Server_Context

Trust Store Configuration

Path: clientStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration

Type: JKS
Path: serverStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced

Externalizing a Mule runtime's certificates

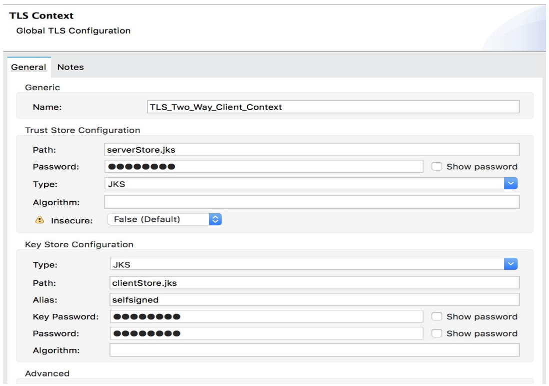
- A customer-hosted Mule runtime can externalize certificates to a folder outside the Mule_HOME location
  - Add a certificate folder in the classpath in the wrapper.conf file in <Mule_HOME>/conf
    - `wrapper.java.classpath.3=%CERT_DIREFCTORY%`
  - Secure %CERT_DIRECTORY% using operating system permissions

Externalizing certificates in Mule runtime with Secret Manager

- **Secret Manager** can be used to store and control access to private keys, passwords, certificates, and other secrets
  - Currently, Secrets Manager is only supported for customer-hosted Mule runtimes in **Runtime Fabric** and **Anypoint Functional Monitoring**
  - On other unsupported deployment platforms (runtime planes) a custom certificate management solution could be used or created

Exercise 16-1: Identify transport layer security for a sample Mule application

- Identify transport layer security for a Mule application
- Identify ways to secure certificates for a Mule application


TLS Context
Global TLS Configuration

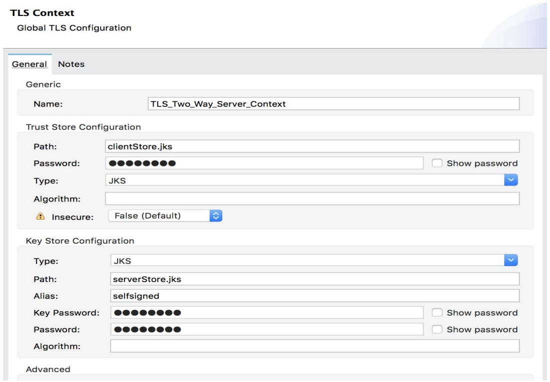
General | Notes

Generic
Name: TLS_Two_Way_Client_Context

Trust Store Configuration
Path: serverStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration
Type: JKS
Path: clientStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced


TLS Context
Global TLS Configuration

General | Notes

Generic
Name: TLS_Two_Way_Server_Context

Trust Store Configuration
Path: clientStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration
Type: JKS
Path: serverStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced

Exercise steps

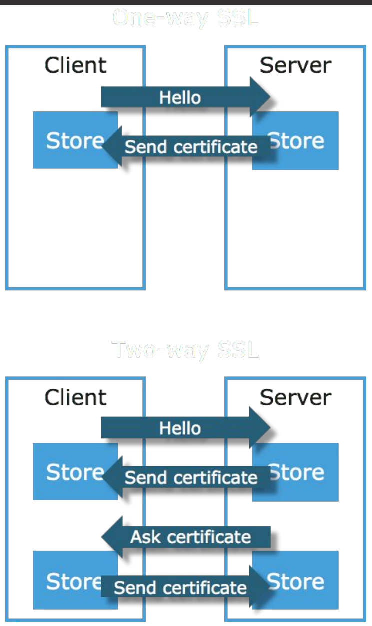
- Identify transport layer security for sample Mule application (security-misconfiguration.jar) of Exercise 16-1 deployed to a customer-hosted runtime plane
- Design ways to secure the Mule application's certificates

Exercise steps

- Identify clients of a Mule application
- Decide whether each client can secure private keys
- Decide the handshake mode between clients and servers (one-way or two-way)
- Identify mechanisms to secure server and client certificates

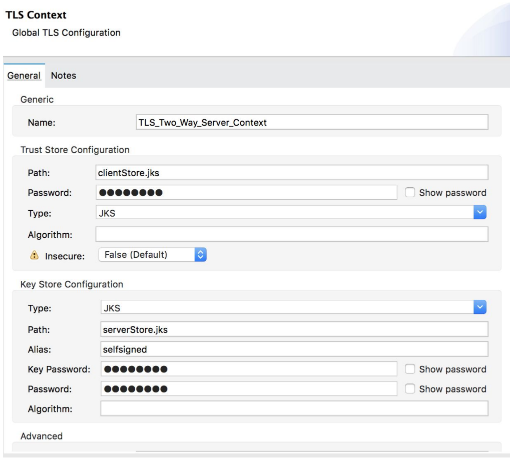
TLS best practices

- If you are authorizing a machine to access resources, it is advisable to have two-way TLS
- A customer-hosted Mule runtime should externalize certificates in a secure external directory



Reflection questions

- Do you use TLS in your Mule apps?
- If so, what issues did you have with TLS?



TLS Context
Global TLS Configuration

Generic

Name: TLS_Two_Way_Server_Context

Trust Store Configuration

Path: clientStore.jks
Password: ●●●●●●●● □ Show password
Type: JKS
Algorithm:
Insecure: False (Default)

Key Store Configuration

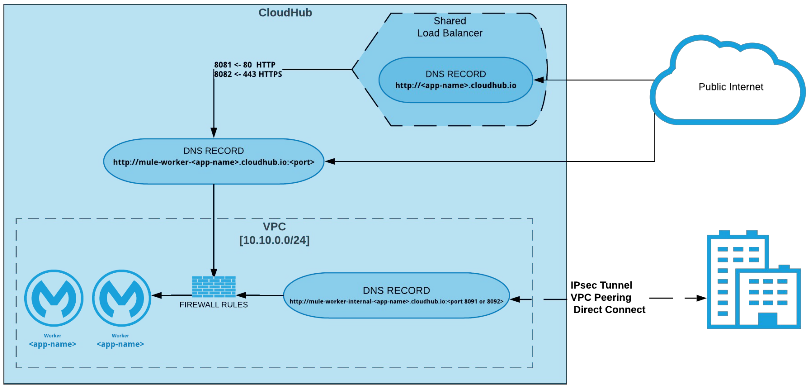
Type: JKS
Path: serverStore.jks
Alias: selfsigned
Key Password: ●●●●●●●● □ Show password
Password: ●●●●●●●● □ Show password
Algorithm:

Advanced

Designing network security options

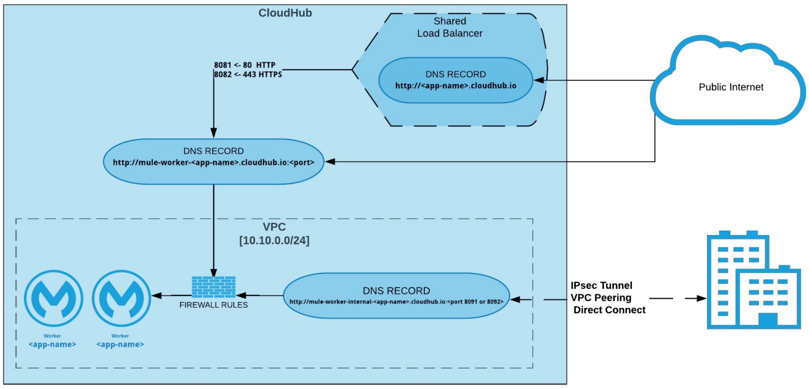
Introducing Anypoint Virtual Private Cloud (VPC)

- Anypoint VPCs can create a private and isolated network in the cloud to host CloudHub workers
- Mule applications deployed to the VPC can communicate with each other using the VPCs private network addresses



Using dedicated load balancers (DLBs) with a VPC

- The VPC can remain completely isolated from external networks
  - But usually a VPC uses one or more dedicated load balancers (DLBs) to route traffic with external public and private networks
  - A DLB can handle Mule event payloads up to 200 MB



Securing network communication through a CloudHub dedicated load balancer (DLB)

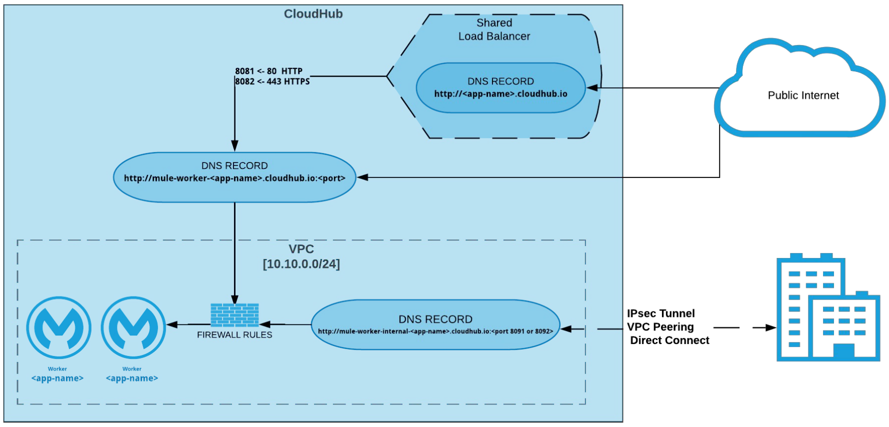
- CloudHub provides a shared load balancer for deployed Mule applications
- Dedicated load balancers can also be purchased and installed into a VPC
- Each DLB is provisioned in AWS and is tied to one CloudHub AWS region
  - Mule applications deployed to an AWS region and environment associated with a VPC and DLB can then receive traffic from the DLB
- Like the shared load balancer service, a DLB
  - Provides a DLB URL to load balance traffic to multiple workers

How a VPC isolated network is typically used

- Host your applications in a VPC and take advantage of DLBs
- Configure customized firewall rules for Mule applications deployed into the VPC
  - Open other ports besides 80, 8081, and 8082
  - Block direct connections to CloudHub workers over port 8081 or 8082
  - Allow or restrict any other TCP or UDP ports
- Connect a VPC to a corporate intranet
  - Whether on-premises or in other clouds
  - Via a VPN connection as if they were all part of a single, private network
- Connect your VPC to another AWS VPC using VPC Peering

Hiding the DLB hostnames

- A DLB's DNS entries are A records and are maintained by MuleSoft
- Customers can define their own "vanity domain names" as CNAMEs (for those A records) in their own DNS servers



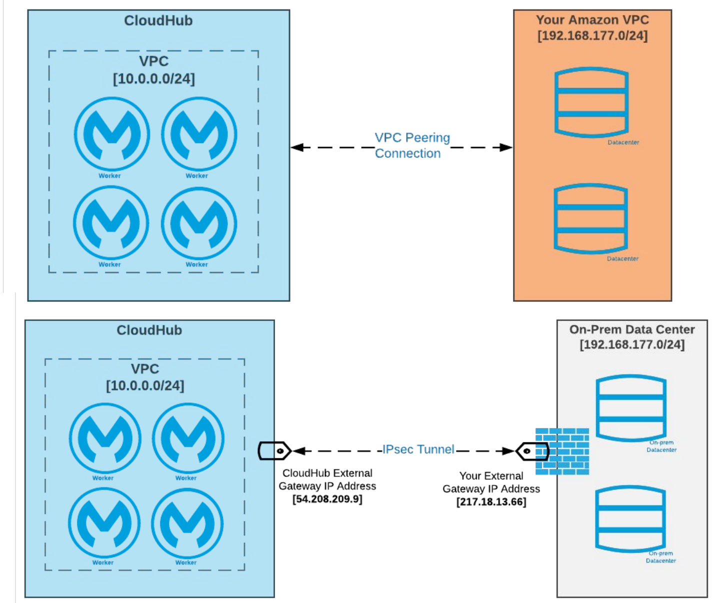
Endpoint security using a CloudHub DLB

- A CloudHub DLB performs TLS termination
  - Just like the CloudHub Shared Load Balancer
- A CloudHub DLB must be configured with server-side certificates for a public-private key pair for the HTTPS endpoints it exposes
  - Optionally, client certificates can be added to a CloudHub DLB so that it performs TLS mutual (two-way) authentication
- The CloudHub DLB can enforce whitelisting of API clients
  - This is often used to define a CloudHub Dedicated Load Balancer for VPC-private APIs
- The DLB receives public IP addresses
- The public IP address' DNS name is an A record
  - So arbitrary CNAMEs for "vanity domain names" can be defined to point to that record

VPC connectivity methods

- Public internet
  - Default connectivity to CloudHub VPC
  - Less reliable as connectivity reliability drive by reliability of ISP
  - Moderate to no security as security is drive by endpoint of resource
- IPSec tunnel with network-to-network configuration
  - An Anypoint VPC can be connected to an on-premises network using an IPsec tunnel
- Amazon Direct Connect
  - An Anypoint VPC can be connected to an on-premises network using an AWS Direct Connect
- VPC Peering
  - Connect an Amazon VPC directly to a Anypoint VPC

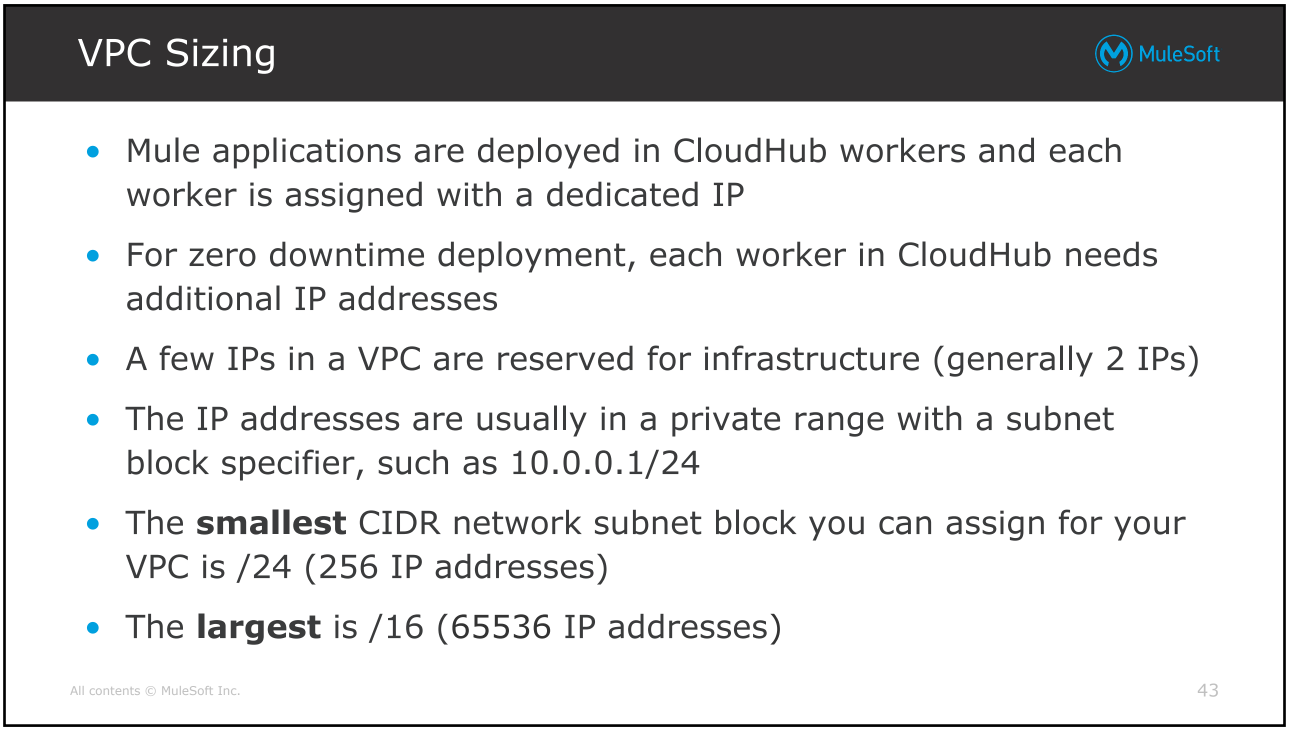
VPC peering vs VPN



VPC peering

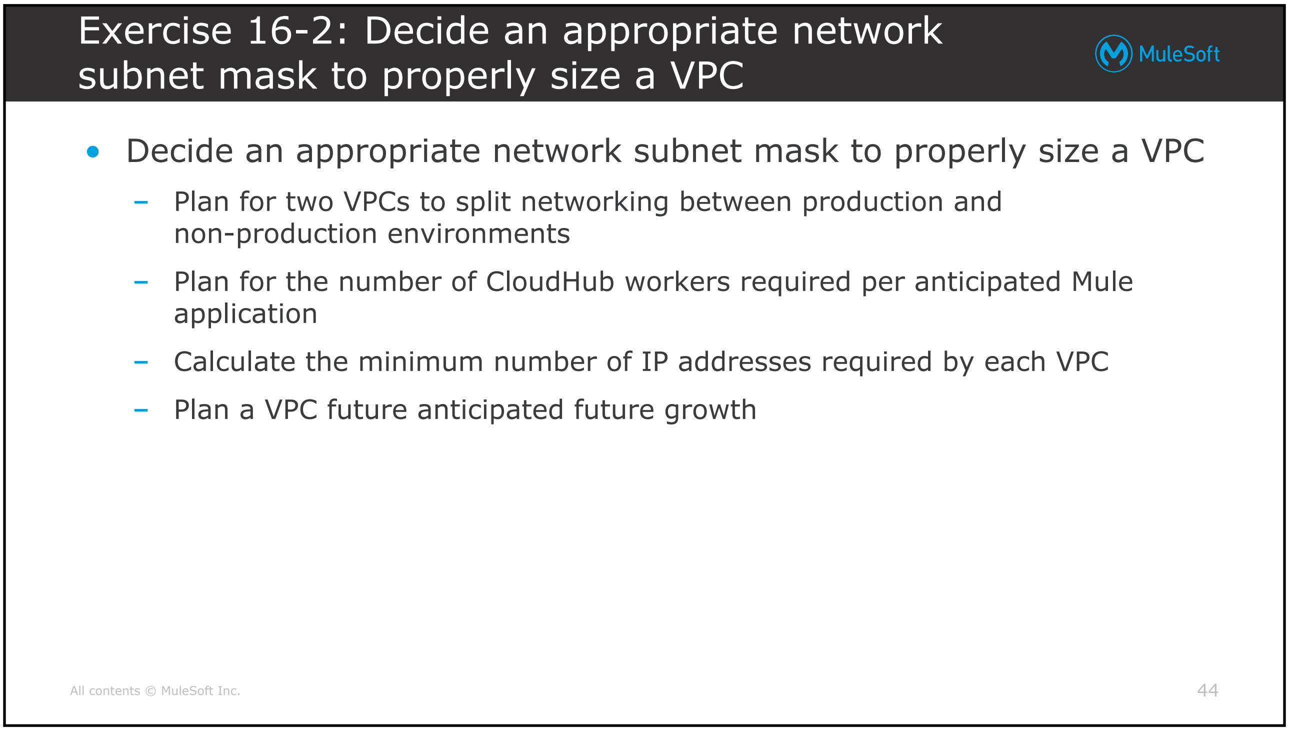
- VPC peering uses **AWS infrastructure** and does not use the internet between regions
- Is the **preferred** way of communicating between an Anypoint VPC and a customer's AWS VPC
- Inter-region traffic is encrypted and intra-region traffic is not encrypted

Properly sizing a VPC to support Mule app deployments



VPC Sizing

- Mule applications are deployed in CloudHub workers and each worker is assigned with a dedicated IP
- For zero downtime deployment, each worker in CloudHub needs additional IP addresses
- A few IPs in a VPC are reserved for infrastructure (generally 2 IPs)
- The IP addresses are usually in a private range with a subnet block specifier, such as 10.0.0.1/24
- The **smallest** CIDR network subnet block you can assign for your VPC is /24 (256 IP addresses)
- The **largest** is /16 (65536 IP addresses)



Exercise 16-2: Decide an appropriate network subnet mask to properly size a VPC

- Decide an appropriate network subnet mask to properly size a VPC
  - Plan for two VPCs to split networking between production and non-production environments
  - Plan for the number of CloudHub workers required per anticipated Mule application
  - Calculate the minimum number of IP addresses required by each VPC
  - Plan a VPC future anticipated future growth

Exercise context

MuleSoft

- The organization has **four** different environments
  - Dev, staging, performance, and production
- The organization is planning to deploy **50** applications in each environment
- The organization is planning to have **two** separate **VPCs** for production and non-production environments
- In the **performance** and **production** environments, each Mule application will be deployed to **two** workers
  - In other environments, the Mule application is only deployed to one worker
- Decide the **minimum** CIDR network subnet for the each of the VPCs (non-production and production environments)

Exercise context: Fill in this table

MuleSoft

- Fill in this table to predict the minimum number of IPs required

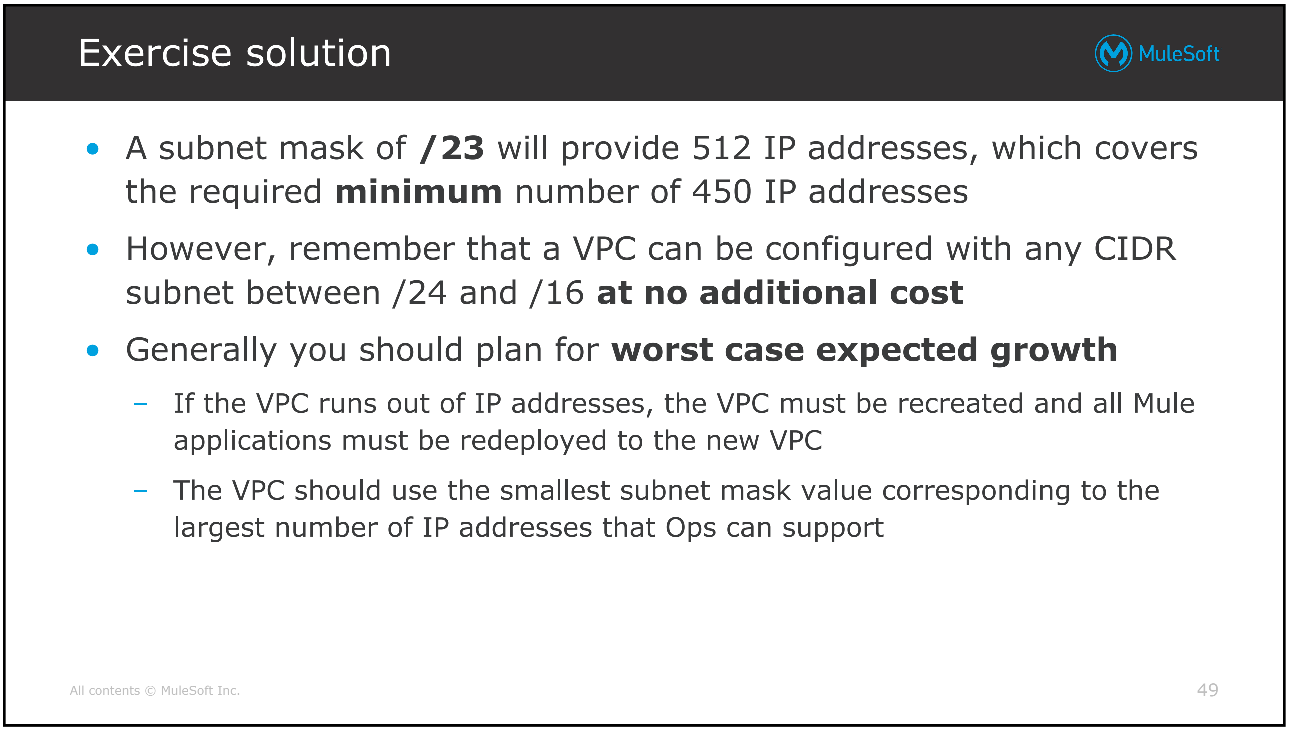
Env	**Production VPC**	**Non-production VPC**
Dev		
Staging		
Performance		
Production		
Total		
Additional IP for zero downtime deployment(50%)		
Total IPs		

Exercise step

- Go to https://www.ipaddressguide.com/cidr and find the CIDR network subnet to support the minimum required total number of IPs

Exercise step

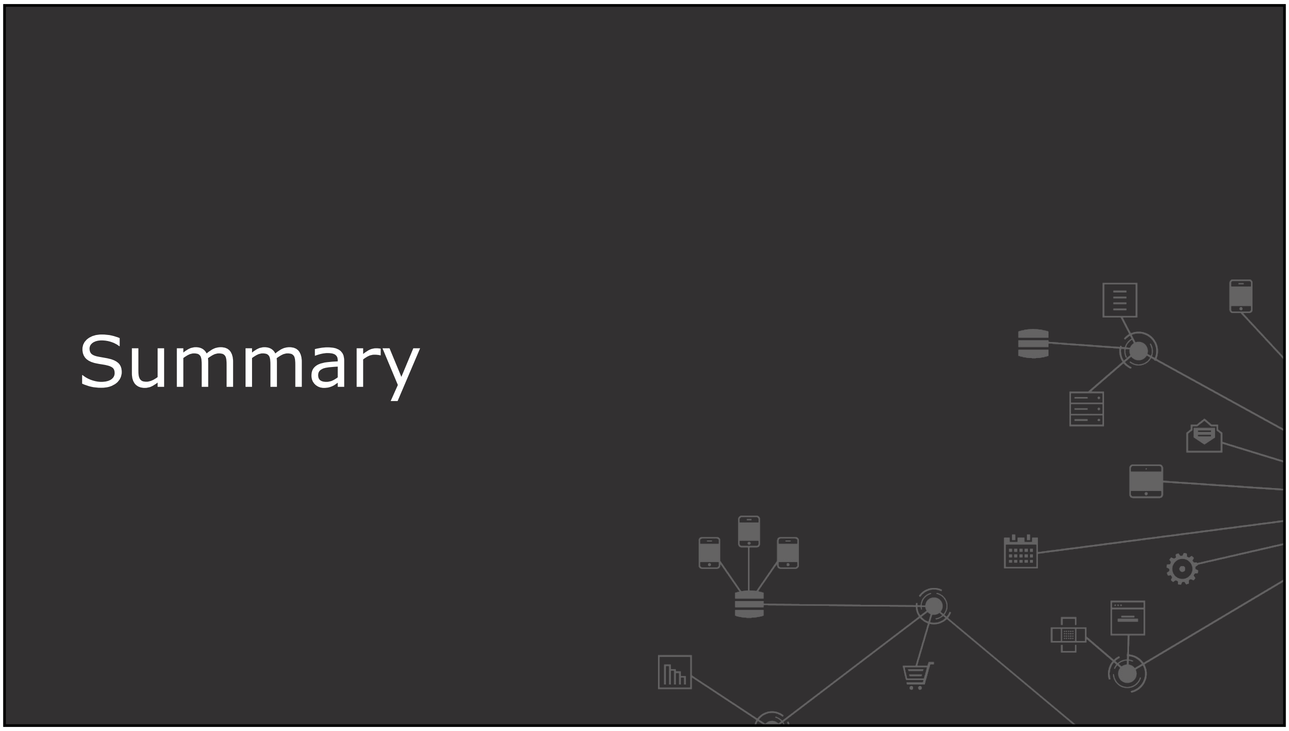
Env	**Production VPC**	**Non-production VPC**
Dev		50
Staging		50
Performance		50 * 2 = 100
Production	50 * 2 = 100	
Total	100	200
Additional IP for zero downtime deployment(50%)	50	100
Total IPs	150	300



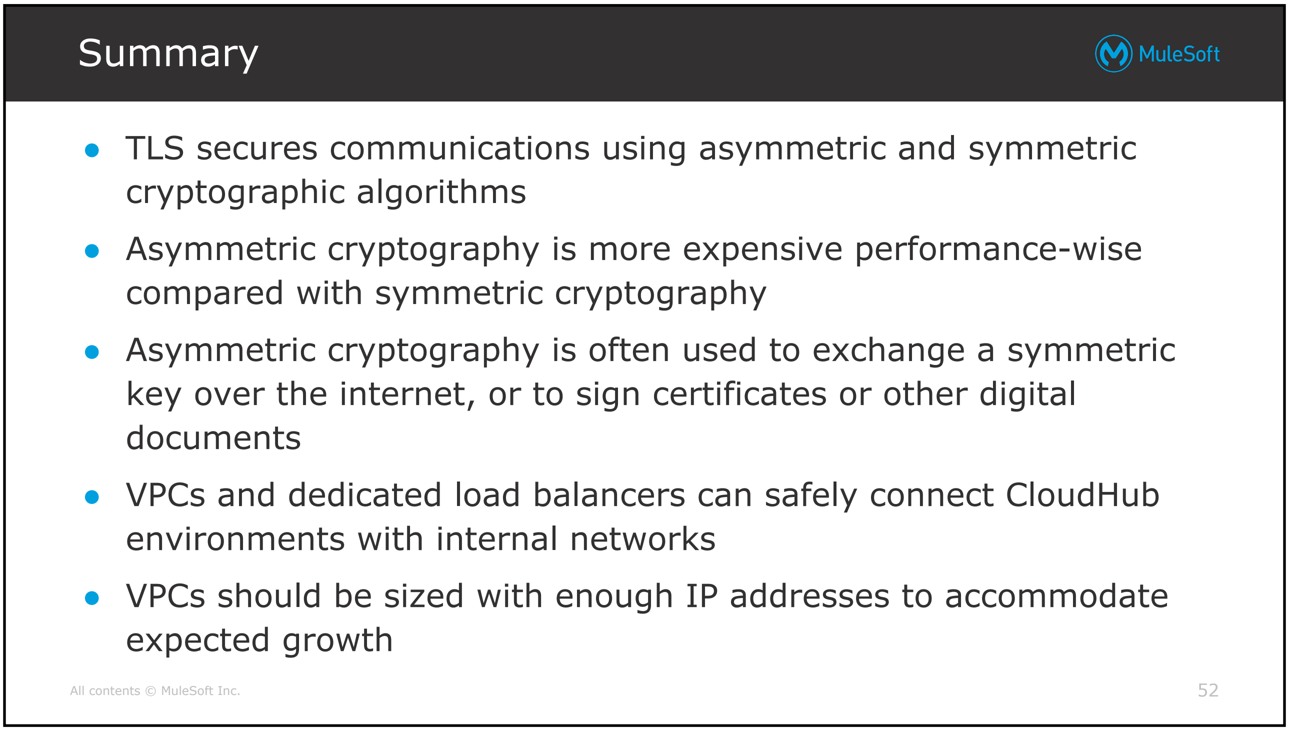
Exercise solution

MuleSoft

- A subnet mask of **/23** will provide 512 IP addresses, which covers the required **minimum** number of 450 IP addresses
- However, remember that a VPC can be configured with any CIDR subnet between /24 and /16 **at no additional cost**
- Generally you should plan for **worst case expected growth**
  - If the VPC runs out of IP addresses, the VPC must be recreated and all Mule applications must be redeployed to the new VPC
  - The VPC should use the smallest subnet mask value corresponding to the largest number of IP addresses that Ops can support



Summary



Summary

MuleSoft

- TLS secures communications using asymmetric and symmetric cryptographic algorithms
- Asymmetric cryptography is more expensive performance-wise compared with symmetric cryptography
- Asymmetric cryptography is often used to exchange a symmetric key over the internet, or to sign certificates or other digital documents
- VPCs and dedicated load balancers can safely connect CloudHub environments with internal networks
- VPCs should be sized with enough IP addresses to accommodate expected growth