PERCONA

TRAINING

Security

http://www.percona.com/training/

Security

AUTHENTICATION OPTIONS

Authentication Options

- Challenge/response authentication using SCRAM-SHA-1 (username & password)
- x.509 Authentication (using Certificates)
- LDAP/ Kerberos
  - Requires paying MongoDB Enterprise subscription
  - Free with Percona Server for MongoDB

User management methods

- db.auth()
- db.changeUserPassword()
- db.createUser()
- db.dropUser()
- db.dropAllUsers()
- db.getUser()
- db.getUsers()

User management methods (2)

- db.grantRolesToUser()
- db.removeUser()
- db.removeRolesFromUser()
- db.updateUser()
- passwordPrompt()

The localhost exception

- Allows you to create a first user by connecting via `localhost`
- Once at least one user has been added, the localhost exception is automatically disabled

Challenge/Response Authentication

- aka Username and Password

```
$ mongod --dbpath ~/test1/ --port 27017 --oplogSize 200 --auth

> use admin
> db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)

# mongo --port 27017 -u "myUserAdmin" -p "abc123" --authenticationDatabase "admin"
```

Certificate-based authentication

- Create the user in `$external` database

```
> db.getSiblingDB("$external").runCommand( {
  createUser: "CN=TEST-CLIENT,OUT=QAClient,O=MongoDB,ST=California,C=US",
  roles: [
    { role: 'readWrite', db: 'test' },
    { role: 'userAdminAnyDatabase', db: 'admin' }
  ]
})

# mongo --tls --tlsCertificateKeyFile /crts/client.pem
db.getSiblingDB("$external").auth( { mechanism: "MONGODB-X509",
 user:"CN=TEST-CLIENT,OUT=QAClient,O=MongoDB,ST=California,C=US"})
```

- Client doesn't have to specify user/pwd

Authentication gotchas

- When adding a user, you create the user in a specific database (e.g. admin)
- This database is the authentication database for the user
- A user created in one database can have permissions to act on other databases

Exercise - Authentication

1. Create a root user via localhost exception and authenticate

```
> use admin
> db.createUser({ user: "root",
  pwd: "percona",
    roles:
    [
     { role: "root", db: "admin" }
  ]
    })

> db.auth("root", "percona")
```

Internal membership authentication

- Replica set members can authenticate via:
  - Shared keyfile
  - Server certificates
- Note: Enabling internal authentication implicitly enables client authorization (e.g. user/pwd)

Exercise - Update replica set to keyfile authentication

1. Stop running mongod process by typing ctrl+C on each terminal

2. Create a keyfile and copy to all members

```
openssl rand -base64 756 > /var/lib/mongo/keyfile
```

3. Give it the proper permissions

```
chmod 400 /var/lib/mongo/keyfile
chown mongod: /var/lib/mongo/keyfile
```

Exercise - Update replica set to keyfile authentication

4. Start the 3 mongod with authentication

```
screen -S rs1 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs1 \
  --port 27017 --oplogSize 200 --wiredTigerCacheSizeGB 0.25 --keyFile /var/lib/mor

screen -S rs2 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs2 \
  --port 27018 --oplogSize 200 --wiredTigerCacheSizeGB 0.25 --keyFile /var/lib/mor

screen -S rs3 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs3 \
  --port 27019 --oplogSize 200 --wiredTigerCacheSizeGB 0.25 --keyFile /var/lib/mor
```

5. Test replica set is functional

```
mongo // connect to the default port 27017
rs.status()
```

Roles

INTRODUCTION

Roles

- Each user has a set of potential roles
  - read, readWrite, dbAdmin, etc.
- Each role applies to one database
  - A single user can have roles on each database
  - Some roles apply to all databases
  - You can also create custom roles

Built-in roles

- read/readWrite
- dbAdmin
- dbOwner
- userAdmin
- *anyDB
- clusterMonitor vs clusterManager
- backup/restore
- root

Roles

RESOURCES

Access Resources

- Database, collection(s) or cluster resource
- Roles define what resource(s) a user has access to
  - a specific collection in a db

```
{ db: <database>, collection: <collection> }
```

  - all collections with specified name across all db

```
{ db: "", collection: <collection> }
```

  - any collection in specified db (excluding `system` collections)

```
{ db: <database>, collection:"" }
```

  - cluster resources

Access Resources

- Actions are the things that can be done (typically used with custom roles, coming up next)

```
{ resource: { <resource> }, actions: [ "<action>", ... ]}
```

- "any resource" , "any action"

https://docs.mongodb.com/manual/reference/privilege-actions/

Exercise - roles

1. Check out existing built-in roles

```
>  db.getRoles( { showBuiltinRoles: true } )
```

2. Check out built-in roles with their associated privileges

```
>  db.getRoles( { showBuiltinRoles: true, showPrivileges: true} )
```

3. Create a read-only user for the training database

Exercise - roles

```
db.createUser( { user: "readonly_user", pwd: "123456", roles: [ { role: "read", db:"training" }] } );
```

Roles

CUSTOM ROLES

Custom Roles

- Same as built-in roles, can be added to `admin` or specific database
- Can inherit from built-in and other custom roles
- Collection is the smallest unit of resource to assign an action

Custom Roles

```
use admin
db.createRole({
  role: "myClusterwideAdmin",
  privileges: [
    { resource: { cluster: true }, actions: [ "addShard" ] },
    {
      resource: { db: "config", collection: "" },
      actions: [ "find", "update", "insert", "remove" ]
    },
    {
      resource: { db: "users", collection: "usersCollection" },
      actions: [ "update", "insert", "remove" ] },
    {
      resource: { db: "", collection: "" },
      actions: [ "find" ]
    } ],
  roles: [ { role: "read", db: "admin" } ]
  },
  { w: "majority" , wtimeout: 5000 }
)
```

Exercises - Custom Roles

1. Create a custom role for Percona Backup

```
> db.getSiblingDB("admin").createRole({ "role": "pbmAnyAction",
      "privileges": [
         { "resource": { "anyResource": true },
           "actions": [ "anyAction" ]
         }
      ],
      "roles": []
   });
```

Exercises - Custom Roles

2. Create the PBM user and grant the required roles

```
> db.getSiblingDB("admin").createUser({user: "pbmuser",
      "pwd": "secretpwd",
      "roles" : [
         { "db" : "admin", "role" : "readWrite", "collection": "" },
         { "db" : "admin", "role" : "backup" },
         { "db" : "admin", "role" : "clusterMonitor" },
         { "db" : "admin", "role" : "restore" },
         { "db" : "admin", "role" : "pbmAnyAction" }
      ]
   });
```

Exercises - Custom Roles

3. Verify existing custom roles

```
db.getRoles( { showPrivileges:true } )
```

4. Verify existing users

```
db.system.users.find().pretty()
```

Security

EXTERNAL AUTHENTICATION

External Authentication

- Allows using external authentication service like Active Directory or OpenLDAP
  - Remotely stores user credentials (i.e. user name, password)
- Requires Enterprise subscription from MongoDB Inc.
  - Available as open source, free, feature in Percona Server for MongoDB
  - External users cannot have roles assigned in the admin database

External Authentication options

- Set up SASL daemon
  - acts as server-local proxy between `mongod` and remote LDAP/AD service
  - SASL Library used by `mongod` to talk to SASL daemon
- Direct binding to LDAP
  - doesn't require setting up external components
- Kerberos

Creating and Authenticating Users

- When properly configured, create users with

```
db.getSiblingDB("$external").createUser( {
  user : christian, roles: [ {role: "read", db: "test"} ]
} );
```

- Authenticate with the `auth` command

```
db.getSiblingDB("$external").auth({
  mechanism:"PLAIN", user:"christian", pwd:"secret", digestPassword:false
})
```

- Also

```
db.auth({
  mechanism:"PLAIN", user:"christian", pwd:"secret", digestPassword:false
})
```

Security

AUDIT LOGGING

Audit Logging

- Requires Enterprise subscription with MongoDB Inc.
- Available in Percona Server for MongoDB, free and open source
- Can log to a file or syslog
- Can filter using regexp or standard query selectors (`$eq`, `$in`, etc.)

Audit Logging - Example

Log only events from a user named `tim` and write them to a JSON file

```
mongod \
 --dbpath data/db
 --auditDestination file \
 --auditFormat JSON \
 --auditPath /var/log/psmdb/audit.json \
 --auditFilter '{ "users.user" : "tim" }'

storage:
  dbPath: data/db
auditLog:
  destination: file
  format: JSON
  path: /var/log/psmdb/audit.json
  filter: '{ "users.user" : "tim" }'
```

Security

ENCRYPTION

Encryption at Rest

- WiredTiger encryption available with Enterprise subscription with MongoDB Inc.
  - Free and open-source feature with Percona Server for MongoDB
- Use filesystem encryption as alternative.
  - "Offline" protection only, i.e. hardware theft.

Mongo EE vs PSMDB

- Mongo EE
  - `Localfile`
  - Supports KMIP protocol
  - Supports Amazon KMS
- Percona distribution
  - `Localfile`
  - Supports Hashicorp Vault

Don't use `Localfile` for production deployments

Enabling Encryption at Rest

- Requires empty database
  - use rolling approach
- During the first run generates a secure key and writes it to the vault
- During the subsequent start, the server tries to read the master key from the vault

Encryption at Rest - Percona Server

Add the following to mongod.conf

```
security:
   enableEncryption: true
   vault:
      serverName: 10.0.2.15
      port: 8200
      secret: secret/data/dc/psmongodb1
      tokenFile: /etc/mongodb/token
      serverCAFile: /etc/mongodb/vault.crt
```

Key rotation

- Rolling approach
- Rotating the master key process re-encrypts the keystore using the new master key
- The new master key is stored in the vault
- The entire dataset is not re-encrypted

Encryption

TLS ENCRYPTION

`mongod` & `mongos` with SSL/TLS

- SSL options were deprecated in 4.2, use TLS instead

1. TLS key file (`--tlsCertificateKeyfile`)

- Generated using the `openssl` command or by SSL provider

2. TLS `mode` (`--tlsMode`)

- `allowTLS` - Allow mixed conns, TLS or plain-text
- `preferTLS` - Allow mixed conns, TLS (prefer) or plain-text
- `requireTLS` - Require TLS on ALL connections

mongod & mongos with SSL/TLS

3. (Optional) TLS CA certificate for validation (`--tlsCAFile`)

4. (Optional) TLS CA certificate revoke list (`--tlsCRLFile`)

TLS with `mongo` Shell

- TLS key file (`--tlsCertificateKeyfile`)
  - Generated using the `openssl` command or by SSL provider
- (Optional) TLS CA certificate for validation (`--tlsCAFile`)
- (Optional) TLS CA certificate revoke list (`--tlsCRLFile`)

Additional Optional TLS Parameters

- `--tlslAllowInvalidHostnames`
- `--tlsAllowInvalidCertificates` (use with self-signed)
- `--tlsCertificateKeyFilePassword`

Using x.509 certificates for membership authentication

1. Change auth mode

```
security:
   clusterAuthMode: x509
```

2. specify TLS parameters

```
net:
   tls:
      mode: requireTLS
      certificateKeyFile: <path to its TLS/SSL certificate and key file>
      CAFile: <path to root CA PEM file to verify received certificate>
```

3. (Optional) use a different certificate for replica set communication (`clusterFile`)

- Uses `tlsCertificateKeyfile` otherwise

Encryption

SELF SIGNED SSL CERTIFICATES

Generate Self Signed Certificates

- `openssl` tool can be used to generate certificate.
  - PEM is a combination of certificate and key.

```
$ cd /etc/ssl/
$ openssl req -newkey rsa:2048 -new -x509 -days 365 -nodes \
    -out mongodb-cert.crt -keyout mongodb-cert.key
$ cat mongodb-cert.key mongodb-cert.crt > mongodb.pem
$ chmod 600 mongodb.pem
$ chown mongodb.root mongodb.pem
```

- Configure MongoDB and Restart.

```
net:
   ssl:
      mode: requireTLS
      CertificateKeyfile: /etc/ssl/mongodb.pem
```

Using Self Signed Certificates

- Test connection with `mongo` shell.

```
mongo --tls --tlsCertificateKeyfile /etc/ssl/mongodb.pem \
    --tlsAllowInvalidCertificates --port 27017
```

Your Own CA (Certificate Authority)

- An SSL CA allows the creation of several SSL certificates that are signed by a single `root` key.
- SSL connections must involve certificates signed by the defined CA cert when (`--tlsCAFile <file>`) is passed to mongo, mongod or mongos.

Your Own CA (Certificate Authority)

- Optional feature: certificate revocation. This provides the ability to `ban` a certificate based on a blacklist file. This is enabled by (`--tlsCRLFile <crl file>`).
- Important:
  - Store the CA key carefully
  - use safe file privileges on certs!

Self Signed CA and Certificates

- Generate CA key

```
openssl genrsa -des3 -out mongodb-ca.key 2048
```

- Generate the CA certificate

```
openssl req -x509 -new -nodes -key mongodb-ca.key -sha256 \
    -days 1024 -out mongodb-ca.pem
```

- Generate a Server Key (for each Host)

```
openssl genrsa -out nodeNN.key 2048
```

Self Signed CA and Certificates

- Generate a Server Certificate Signing Request (for each host)

```
openssl req -new -key nodeNN.key -out nodeNN.csr
```

- Sign the Server CSR (for each host)

```
openssl x509 -req -in nodeNN.csr -CA mongodb-ca.pem -CAkey mongodb-ca.key \
    -CAcreateserial -out nodeNN.crt -days 500 -sha256
```

- Create PEM file with proper format

```
cat nodeNN.key nodeNN.crt > nodeNN.pem
```

Self Signed CA and Certificates

- Configure MongoDB and Restart.

```
net:
   ssl:
      mode: requireTLS
      certificateKeyFile: /etc/ssl/nodeNN.pem
      CAFile: /etc/ssl/mongodb-ca.pem
```

- Test Connection

```
mongo --tls --certificateKeyFile /etc/ssl/nodeNN.pem \
    --tlsCAFile /etc/ssh/mongodb-ca.pem --port 27019 admin
```

Security

NETWORK

Network Exposure Options

- `bindIp` limits the IP addresses the server listens on
- Using a non-standard port can provide a layer of obscurity
- MongoDB should still be run only in a trusted environment

Lab Exercises

Self Signed Certificates with CA

Self Signed Certificates with CA

1. Generate CA key

```
openssl genrsa -des3 -out /mongodb/data/mongo-ca.key 2048
```

2. Generate the CA certificate

```
openssl req -x509 -new -nodes -key /mongodb/data/mongo-ca.key -sha256 \
    -days 1024 -out /mongodb/data/mongo-ca.pem
```

3. Generate a Server Key (for each Host)

```
openssl genrsa -out /mongodb/data/rs1.key 2048
openssl genrsa -out /mongodb/data/rs2.key 2048
openssl genrsa -out /mongodb/data/rs3.key 2048
```

Self Signed Certificates with CA

4. Generate a Server Certificate Signing Request (for each Host). Make sure that the value for `Common Name` when prompted, is the value of `localhost` in this case i.e.

```
...
Common Name (e.g. server FQDN or YOUR name) []:localhost
...

openssl req -new -key /mongodb/data/rs1.key -out /mongodb/data/rs1.csr
openssl req -new -key /mongodb/data/rs2.key -out /mongodb/data/rs2.csr
openssl req -new -key /mongodb/data/rs3.key -out /mongodb/data/rs3.csr
```

Self Signed Certificates with CA

5. Sign the Server CSR (for each Host)

```
openssl x509 -req -in /mongodb/data/rs1.csr -CA /mongodb/data/mongo-ca.pem \
  -CAkey /mongodb/data/mongo-ca.key -CAcreateserial \
  -out /mongodb/data/rs1.crt -days 500 -sha256

openssl x509 -req -in /mongodb/data/rs2.csr -CA /mongodb/data/mongo-ca.pem \
  -CAkey /mongodb/data/mongo-ca.key -CAcreateserial \
  -out /mongodb/data/rs2.crt -days 500 -sha256

openssl x509 -req -in /mongodb/data/rs3.csr -CA /mongodb/data/mongo-ca.pem \
  -CAkey /mongodb/data/mongo-ca.key -CAcreateserial \
  -out /mongodb/data/rs3.crt -days 500 -sha256

cat /mongodb/data/rs1.crt /mongodb/data/rs1.key > /mongodb/data/rs1.pem
cat /mongodb/data/rs2.crt /mongodb/data/rs2.key > /mongodb/data/rs2.pem
cat /mongodb/data/rs3.crt /mongodb/data/rs3.key > /mongodb/data/rs3.pem
```

Migrate to SSL/TLS

1. Login to the arbiter, shut it down, and start it back up

```
mongo --port 27019

use admin
db.shutdownServer()

screen -S rs3 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs3 \
  --port 27019 --oplogSize 200 --wiredTigerCacheSizeGB 0.25 --auth \
  --tlsMode allowTLS --tlsCertificateKeyFile /mongodb/data/rs3.pem \
  --tlsCAFile /mongodb/data/mongo-ca.pem --transitionToAuth \
  --clusterAuthMode x509 --tlsClusterFile /mongodb/data/rs3.pem
```

Migrate to SSL/TLS

2. Repeat the process with the other `SECONDARY` node

```
mongo --port 27018

use admin
db.shutdownServer()

screen -S rs2 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs2 \
  --port 27018 --oplogSize 200 --wiredTigerCacheSizeGB 0.25 --auth \
  --tlsMode allowTLS --tlsCertificateKeyFile /mongodb/data/rs2.pem \
  --tlsCAFile /mongodb/data/mongo-ca.pem --transitionToAuth \
  --clusterAuthMode x509 --tlsClusterFile /mongodb/data/rs2.pem
```

Migrate to SSL/TLS

3. Step down the current `PRIMARY`, and repeat the process

```
mongo --port 27017

use admin
rs.stepDown(60)
# use rs.status() to monitor until a new PRIMARY is elected
db.shutdownServer()

screen -S rs1 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs1 \
  --port 27017 --oplogSize 200 --wiredTigerCacheSizeGB 0.25 --auth \
  --tlsMode allowTLS --tlsCertificateKeyFile /mongodb/data/rs1.pem \
  --tlsCAFile /mongodb/data/mongo-ca.pem --transitionToAuth \
  --clusterAuthMode x509 --tlsClusterFile /mongodb/data/rs1.pem
```

Migrate to SSL/TLS

4. Check the console logs of any instance to see if the certificate is being used

```
screen -ls
There are screens on:
  8719.rs1  (Detached)
  8428.rs2  (Detached)
  8370.rs3  (Detached)
3 Sockets in /var/run/screen/S-root.

screen -x 8719.rs1
```

PERCONA
TRAINING

Questions