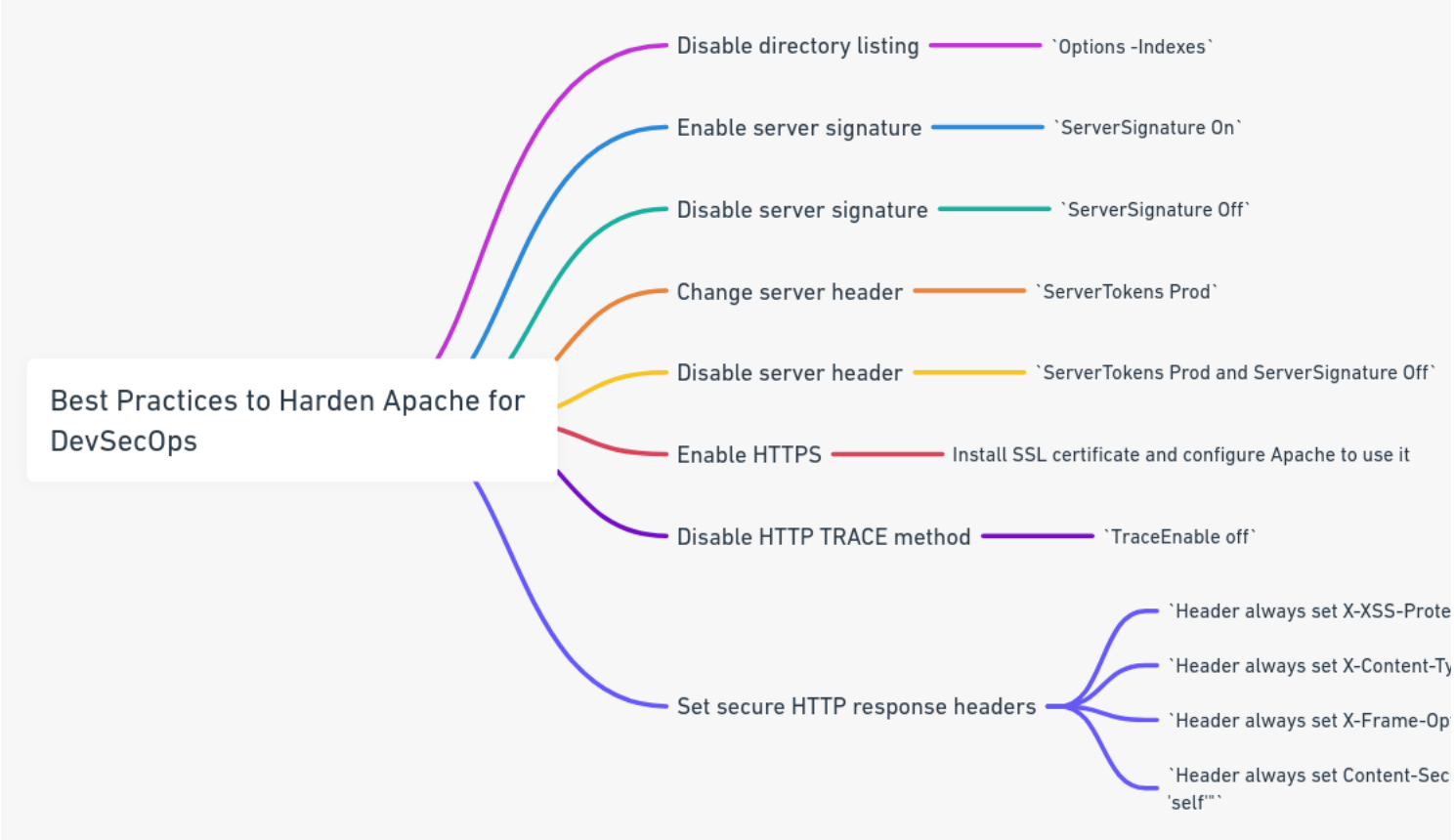

DEVSECOPS
CHECKLIST
PART1
HADESS
WWW.HADESS.IO


Best Practices to Harden Apache for DevSecOps
Disable directory listing
`Options -Indexes`
Enable server signature
`ServerSignature On`
Disable server signature
`ServerSignature Off`
Change server header
`ServerTokens Prod`
Disable server header
`ServerTokens Prod and ServerSignature Off`
Enable HTTPS
Install SSL certificate and configure Apache to use it
Disable HTTP TRACE method
`TraceEnable off`
Set secure HTTP response headers
`Header always set X-XSS-Prote
`Header always set X-Content-Ty
`Header always set X-Frame-Op
`Header always set Content-Sec
'self'"`

Apache Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Apache for DevSecOps

Disable directory listing

```
Options -Indexes
```

Enable server signature

```
ServerSignature On
```

Disable server signature

```
ServerSignature Off
```

Change server header

```
ServerTokens Prod
```

Disable server header

`ServerTokens Prod` and `ServerSignature Off`

Enable HTTPS

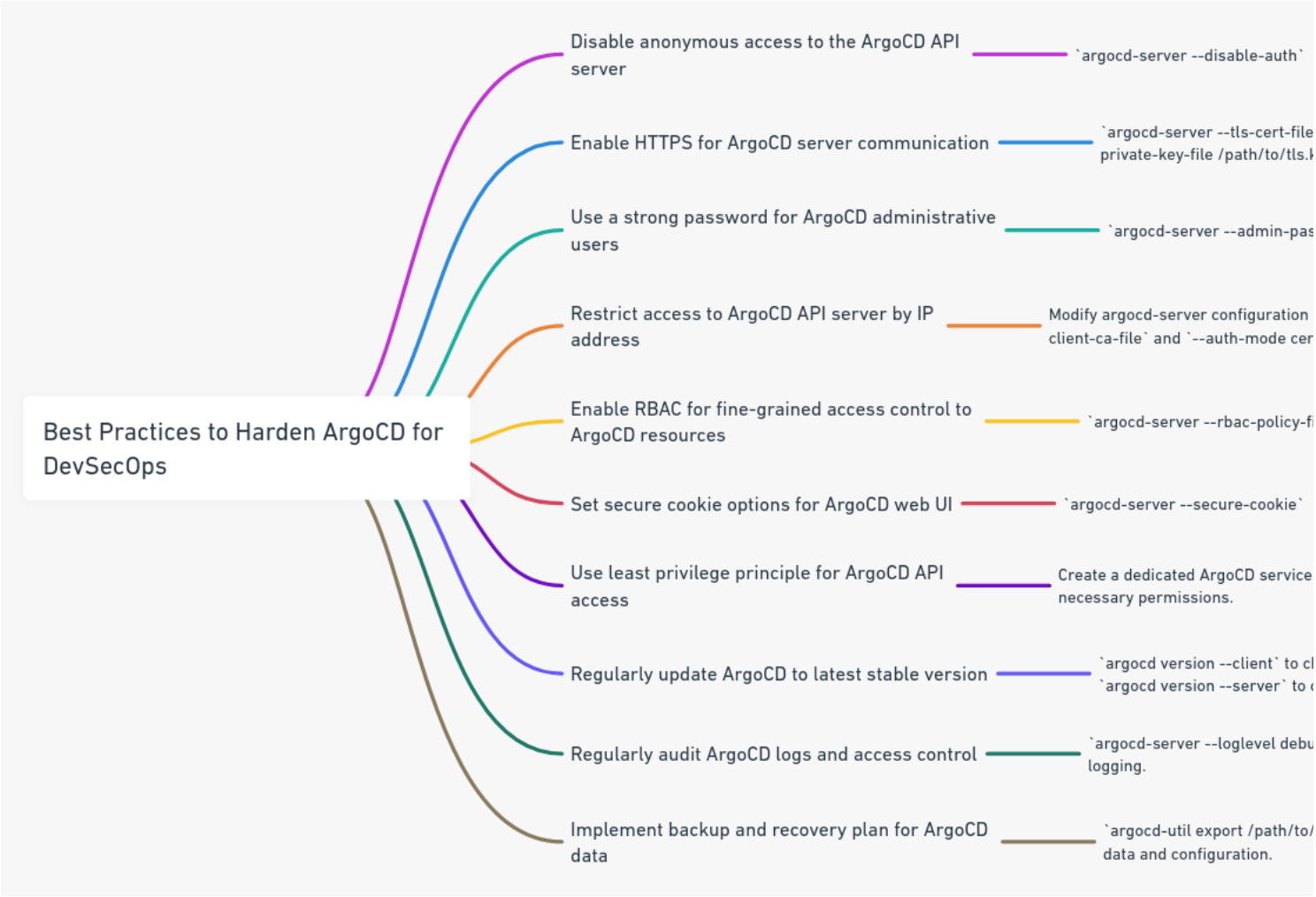
Install SSL certificate and configure Apache to use it

Disable HTTP TRACE method

```
TraceEnable off
```

Set secure HTTP response headers

```
  Header always set X-XSS-Protection "1; mode=block"
  Header always set X-Content-Type-Options nosniff
  Header always set X-Frame-Options SAMEORIGIN
  Header always set Content-Security-Policy "default-src 'self'"
```


Best Practices to Harden ArgoCD for DevSecOps
Disable anonymous access to the ArgoCD API server
`argocd-server --disable-auth`
Enable HTTPS for ArgoCD server communication
`argocd-server --tls-cert-file
private-key-file /path/to/tls.k
Use a strong password for ArgoCD administrative users
`argocd-server --admin-pas
Restrict access to ArgoCD API server by IP address
Modify argocd-server configuration
client-ca-file` and `--auth-mode cer
Enable RBAC for fine-grained access control to ArgoCD resources
`argocd-server --rbac-policy-f
Set secure cookie options for ArgoCD web UI
`argocd-server --secure-cookie`
Use least privilege principle for ArgoCD API access
Create a dedicated ArgoCD service
necessary permissions.
Regularly update ArgoCD to latest stable version
`argocd version --client` to cl
`argocd version --server` to
Regularly audit ArgoCD logs and access control
`argocd-server --loglevel debu
logging.
Implement backup and recovery plan for ArgoCD data
`argocd-util export /path/to/
data and configuration.

ArgoCD Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden ArgoCD for DevSecOps

Disable anonymous access to the ArgoCD API server

```
argocd-server --disable-auth
```

Enable HTTPS for ArgoCD server communication

```
argocd-server --tls-cert-file /path/to/tls.crt --tls-private-key-file /path/to/tls.key
```

Use a strong password for ArgoCD administrative users

```
argocd-server --admin-password <password>
```

Restrict access to ArgoCD API server by IP address

Modify `argocd-server` configuration file to specify `--client-ca-file` and `--auth-mode cert` options and create a certificate authority file and client certificate signed by the CA for each client host.

Enable RBAC for fine-grained access control to ArgoCD resources

```
argocd-server --rbac-policy-file /path/to/rbac.yaml
```

Set secure cookie options for ArgoCD web UI

```
argocd-server --secure-cookie
```

Use least privilege principle for ArgoCD API access

Create a dedicated ArgoCD service account with minimal necessary permissions.

Regularly update ArgoCD to latest stable version

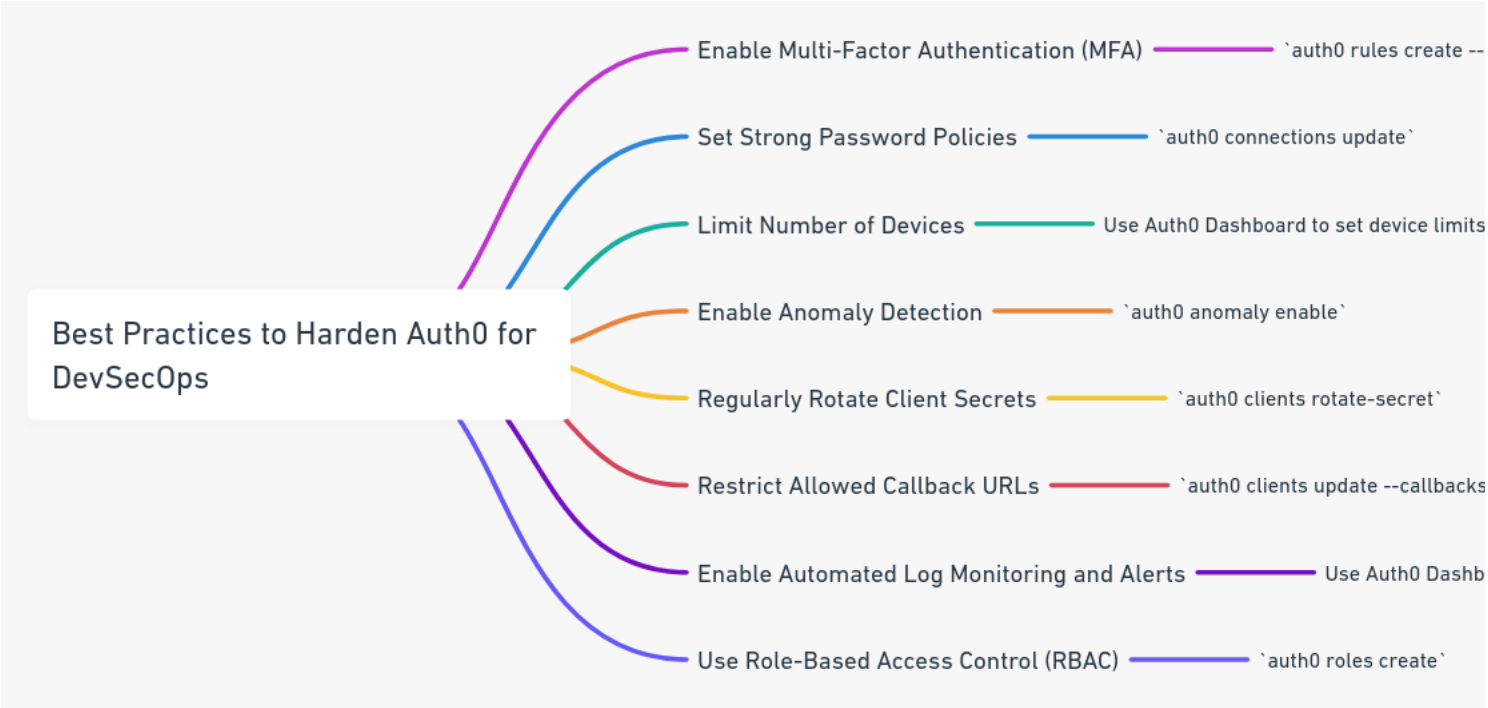
`argocd version --client` to check client version and `argocd version --server` to check server version. Use package manager or manual upgrade as needed.

Regularly audit ArgoCD logs and access control

`argocd-server --loglevel debug` to enable debug level logging. Use a log analyzer or SIEM tool to monitor logs for anomalies.

Implement backup and recovery plan for ArgoCD data

`argocd-util export /path/to/export` to export ArgoCD data and configuration. Store backups securely and test restoration procedure periodically.


Best Practices to Harden Auth0 for DevSecOps
Enable Multi-Factor Authentication (MFA)
`auth0 rules create --
Set Strong Password Policies
`auth0 connections update`
Limit Number of Devices
Use Auth0 Dashboard to set device limits
Enable Anomaly Detection
`auth0 anomaly enable`
Regularly Rotate Client Secrets
`auth0 clients rotate-secret`
Restrict Allowed Callback URLs
`auth0 clients update --callbacks
Enable Automated Log Monitoring and Alerts
Use Auth0 Dashb
Use Role-Based Access Control (RBAC)
`auth0 roles create`

auth0 Security Checklist for DevSecOps

TABLE OF CONTENTS

List of some best practices to auth0 for DevSecOps

Enable Multi-Factor Authentication (MFA)

```
auth0 rules create --name enable-mfa
```

Set Strong Password Policies

```
auth0 connections update
```

Limit Number of Devices

```
Use Auth0 Dashboard to set device limits
```

Enable Anomaly Detection

```
auth0 anomaly enable
```

Regularly Rotate Client Secrets

```
auth0 clients rotate-secret
```

Restrict Allowed Callback URLs

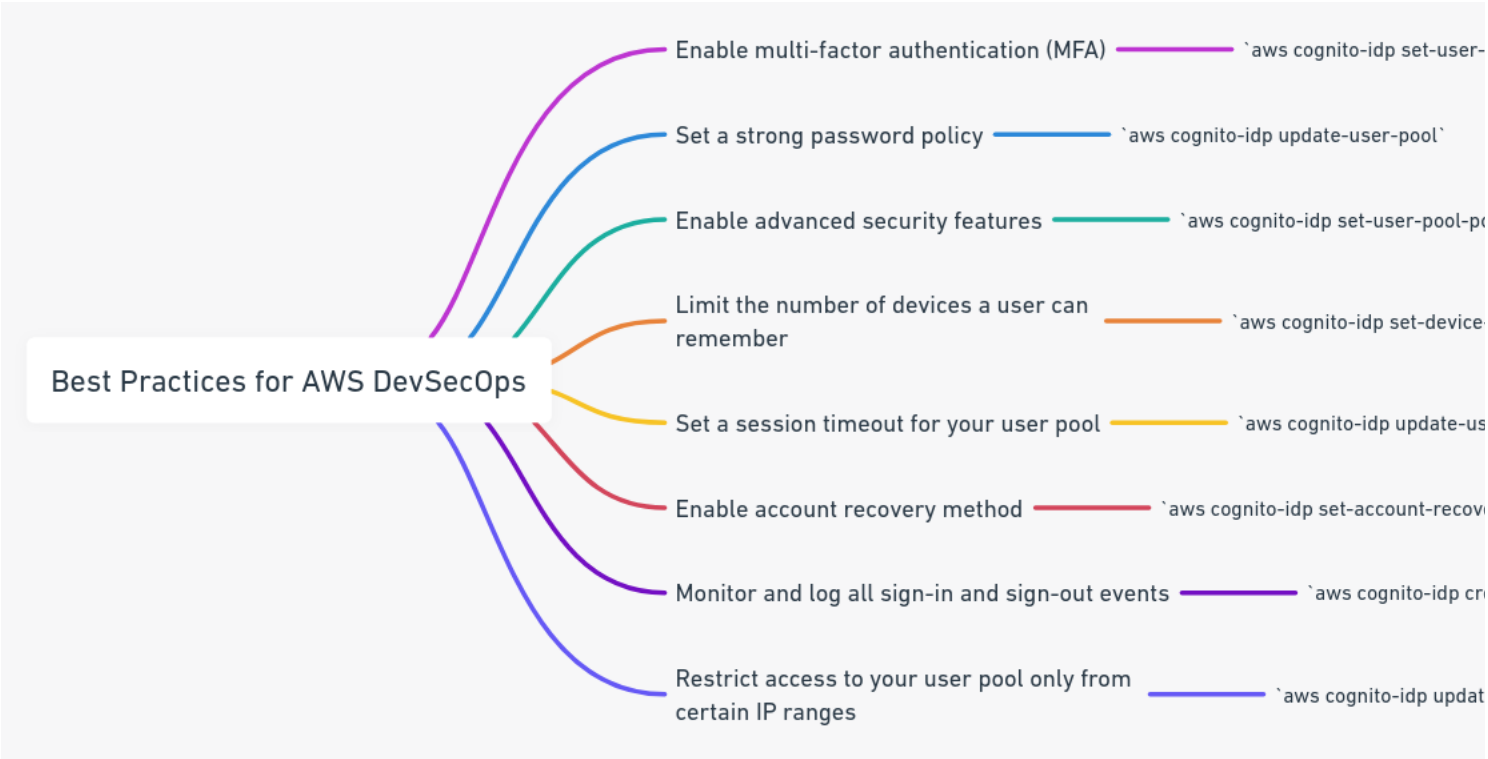
```
auth0 clients update --callbacks
```

Enable Automated Log Monitoring and Alerts

```
Use Auth0 Dashboard to configure alerts
```

Use Role-Based Access Control (RBAC)

```
auth0 roles create
```


Best Practices for AWS DevSecOps
Enable multi-factor authentication (MFA)
`aws cognito-idp set-user-
Set a strong password policy
`aws cognito-idp update-user-pool`
Enable advanced security features
`aws cognito-idp set-user-pool-p
Limit the number of devices a user can remember
`aws cognito-idp set-device
Set a session timeout for your user pool
`aws cognito-idp update-us
Enable account recovery method
`aws cognito-idp set-account-recov
Monitor and log all sign-in and sign-out events
`aws cognito-idp cr
Restrict access to your user pool only from certain IP ranges
`aws cognito-idp updat

AWS Security Checklist for DevSecOps

TABLE OF CONTENTS

List of some best practices to AWS for DevSecOps

Enable multi-factor authentication (MFA)

```
aws cognito-idp set-user-mfa-preference
```

Set a strong password policy

```
aws cognito-idp update-user-pool
```

Enable advanced security features

```
aws cognito-idp set-user-pool-policy
```

Limit the number of devices a user can remember

```
aws cognito-idp set-device-configuration
```

Set a session timeout for your user pool

```
aws cognito-idp update-user-pool-client
```

Enable account recovery method

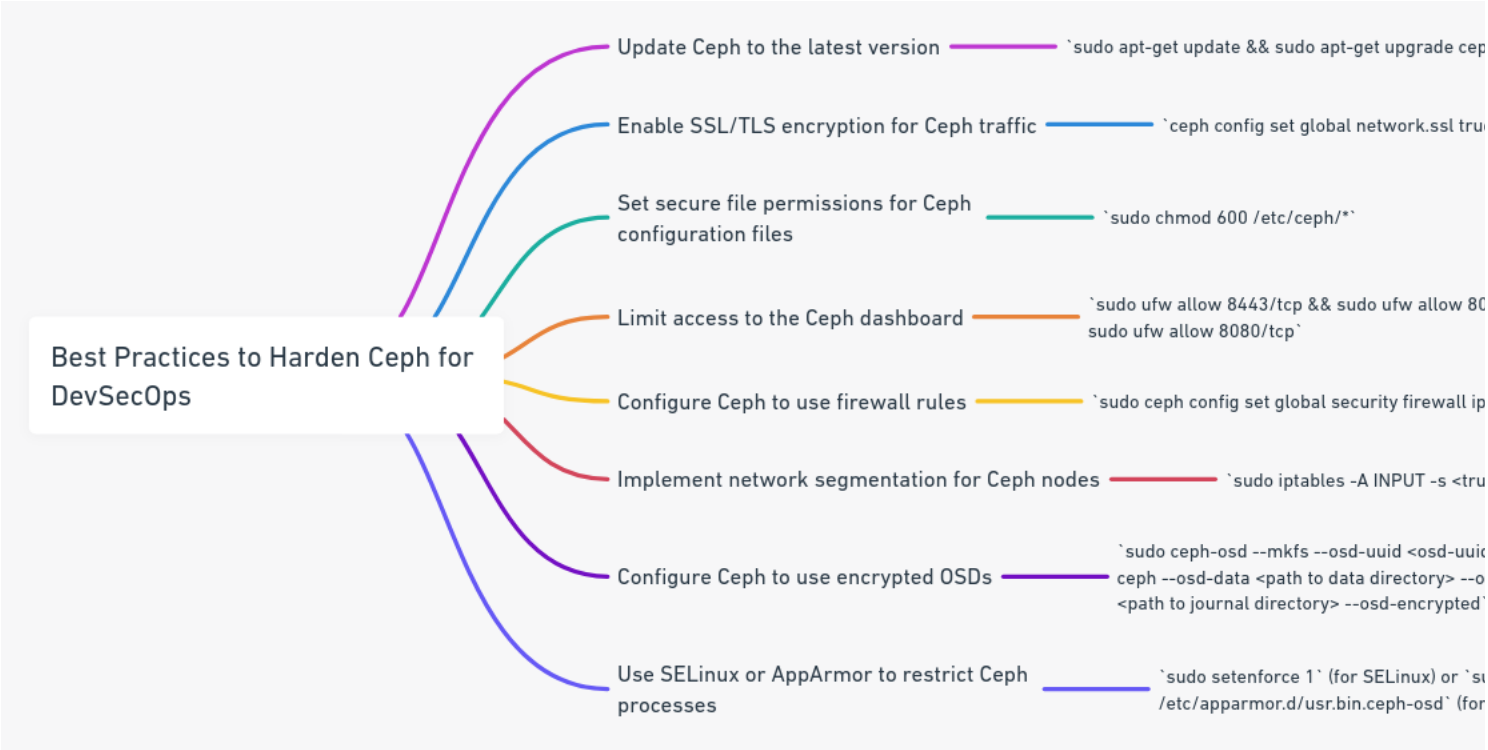
```
aws cognito-idp set-account-recovery
```

Monitor and log all sign-in and sign-out events

```
aws cognito-idp create-user-pool-domain
```

Restrict access to your user pool only from certain IP ranges

```
aws cognito-idp update-resource-server
```


Best Practices to Harden Ceph for DevSecOps
Update Ceph to the latest version
`sudo apt-get update && sudo apt-get upgrade cep
Enable SSL/TLS encryption for Ceph traffic
`ceph config set global network.ssl tru
Set secure file permissions for Ceph configuration files
`sudo chmod 600 /etc/ceph/*`
Limit access to the Ceph dashboard
`sudo ufw allow 8443/tcp && sudo ufw allow 80
sudo ufw allow 8080/tcp`
Configure Ceph to use firewall rules
`sudo ceph config set global security firewall ip
Implement network segmentation for Ceph nodes
`sudo iptables -A INPUT -s <tru
Configure Ceph to use encrypted OSDs
`sudo ceph-osd --mkfs --osd-uuid <osd-uuic
ceph --osd-data <path to data directory> --o
<path to journal directory> --osd-encrypted`
Use SELinux or AppArmor to restrict Ceph processes
`sudo setenforce 1` (for SELinux) or `su
/etc/apparmor.d/usr.bin.ceph-osd` (for

Ceph Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Ceph for DevSecOps

Update Ceph to the latest version

```
sudo apt-get update && sudo apt-get upgrade ceph -y
```

Enable SSL/TLS encryption for Ceph traffic

```
ceph config set global network.ssl true
```

Set secure file permissions for Ceph configuration files

```
sudo chmod 600 /etc/ceph/*
```

Limit access to the Ceph dashboard

```
sudo ufw allow 8443/tcp && sudo ufw allow 8003/tcp && sudo ufw allow 8080/tcp
```

Configure Ceph to use firewall rules

```
sudo ceph config set global security firewall iptables
```

Implement network segmentation for Ceph nodes

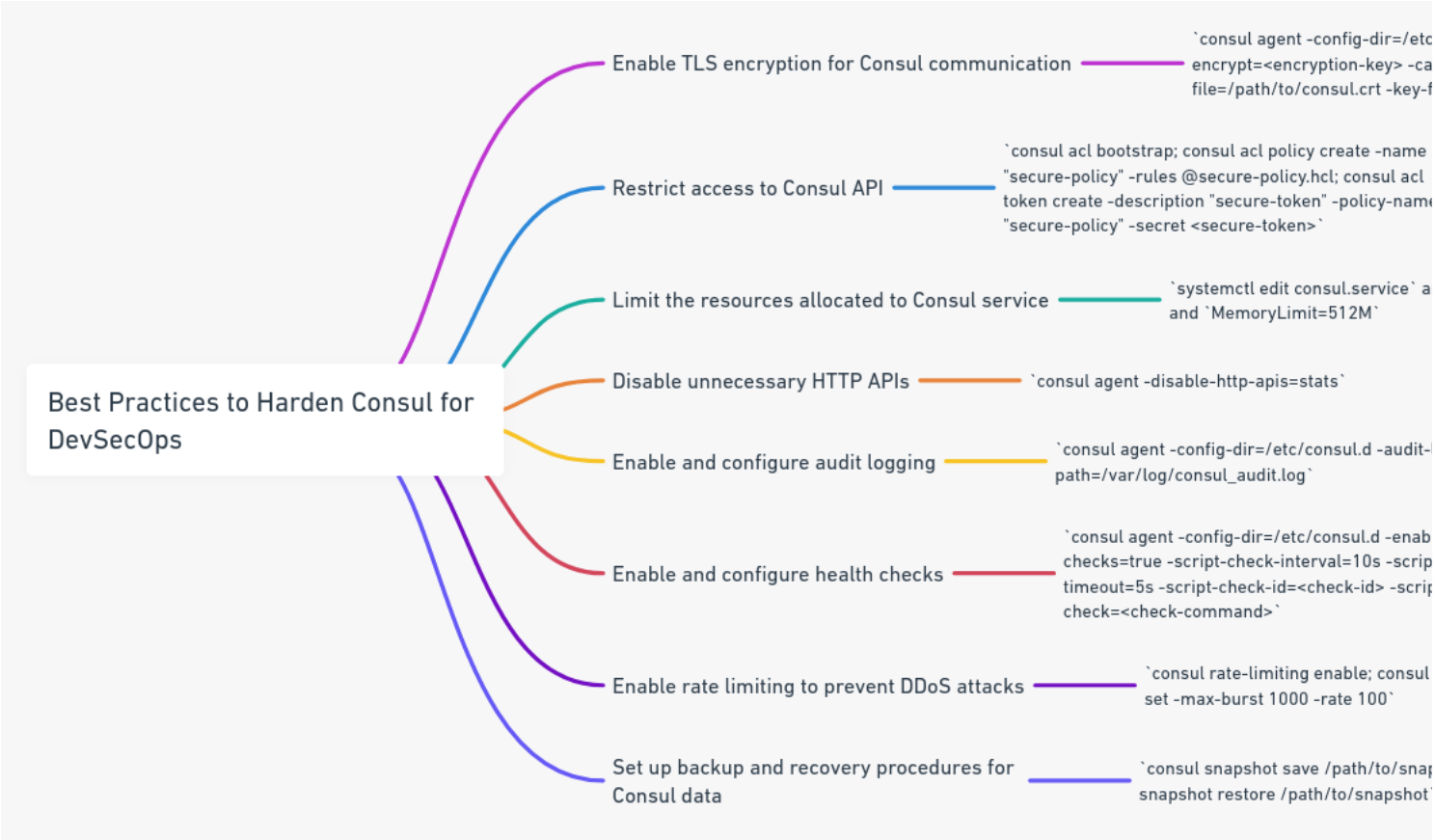
```
sudo iptables -A INPUT -s <trusted network> -j ACCEPT
```

Configure Ceph to use encrypted OSDs

```
sudo ceph-osd --mkfs --osd-uuid <osd-uuid> --cluster ceph --osd-data <path to data directory> --osd-journal <path to journal directory> --osd-encryp
```

Use SELinux or AppArmor to restrict Ceph processes

`sudo setenforce 1` (for SELinux) or `sudo aa-enforce /etc/apparmor.d/usr.bin.ceph-osd` (for AppArmor)


Best Practices to Harden Consul for DevSecOps
Enable TLS encryption for Consul communication
`consul agent -config-dir=/etc
encrypt=<encryption-key> -ca
file=/path/to/consul.crt -key-f
Restrict access to Consul API
`consul acl bootstrap; consul acl policy create -name
"secure-policy" -rules @secure-policy.hcl; consul acl
token create -description "secure-token" -policy-name
"secure-policy" -secret <secure-token>`
Limit the resources allocated to Consul service
`systemctl edit consul.service` a
and `MemoryLimit=512M`
Disable unnecessary HTTP APIs
`consul agent -disable-http-apis=stats`
Enable and configure audit logging
`consul agent -config-dir=/etc/consul.d -audit-
path=/var/log/consul_audit.log`
Enable and configure health checks
`consul agent -config-dir=/etc/consul.d -enab
checks=true -script-check-interval=10s -scrip
timeout=5s -script-check-id=<check-id> -scrip
check=<check-command>`
Enable rate limiting to prevent DDoS attacks
`consul rate-limiting enable; consul
set -max-burst 1000 -rate 100`
Set up backup and recovery procedures for Consul data
`consul snapshot save /path/to/snap
snapshot restore /path/to/snapshot`

Consul Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Consul for DevSecOps

Enable TLS encryption for Consul communication

```
consul agent -config-dir=/etc/consul.d -encrypt=<encryption-key> -ca-file=/path/to/ca.crt -cert-file=/path/to/consul.crt -key-file=/path/to/consul.key
```

Restrict access to Consul API

```
consul acl bootstrap; consul acl policy create -name "secure-policy" -rules @secure-policy.hcl; consul acl token create -description "secure-token" -policy-name "secure-policy" -secret <secure-token>
```

Limit the resources allocated to Consul service

`systemctl edit consul.service` and add `CPUQuota=50%` and `MemoryLimit=512M`

Disable unnecessary HTTP APIs

```
consul agent -disable-http-apis=stats
```

Enable and configure audit logging

```
consul agent -config-dir=/etc/consul.d -audit-log-path=/var/log/consul_audit.log
```

Enable and configure health checks

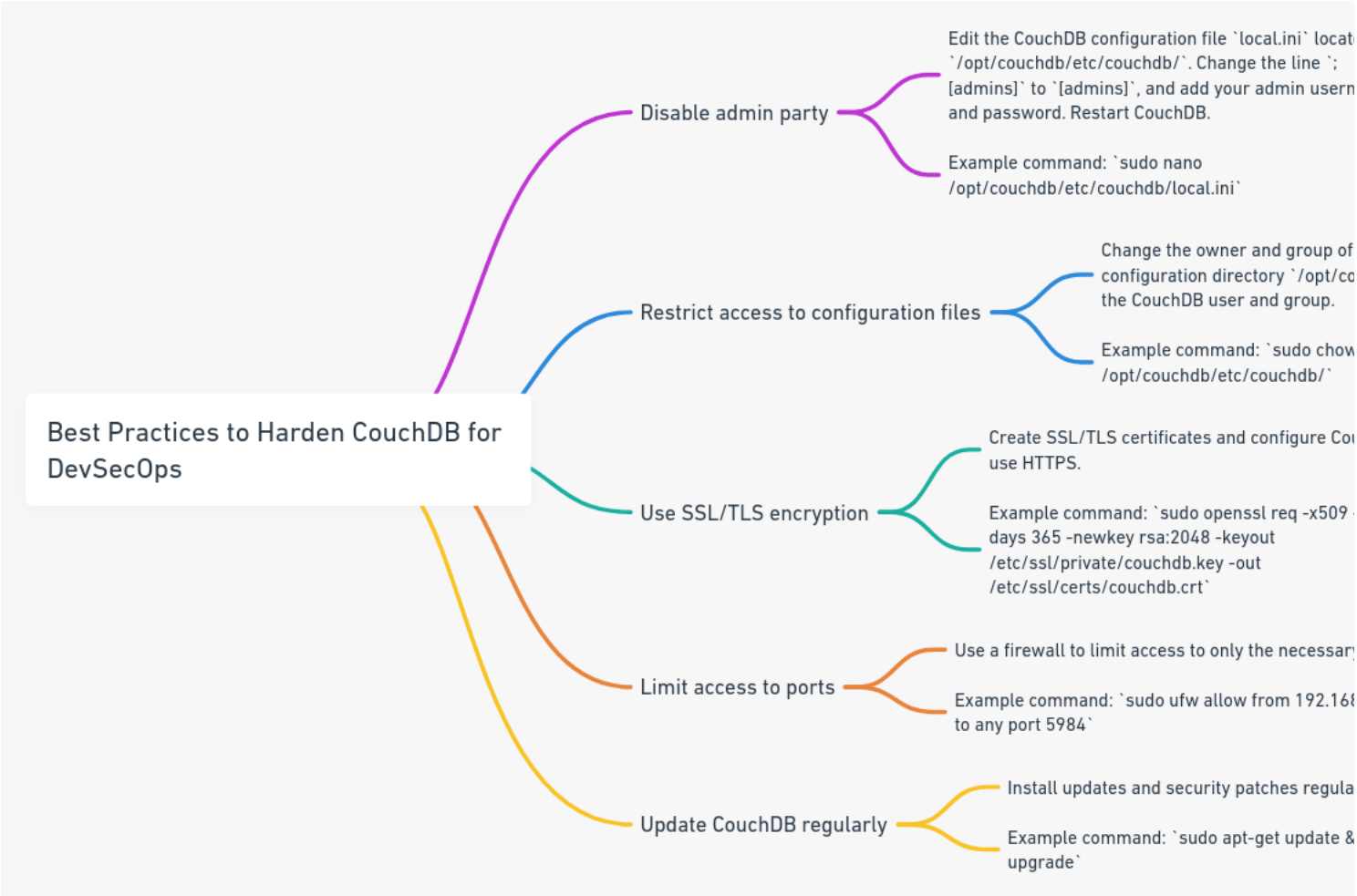
```
consul agent -config-dir=/etc/consul.d -enable-script-checks=true -script-check-interval=10s -script-check-timeout=5s -script-check-id=<check-id> -script-check=<check-command>
```

Enable rate limiting to prevent DDoS attacks

```
consul rate-limiting enable; consul rate-limiting config set -max-burst 1000 -rate 100
```

Set up backup and recovery procedures for Consul data

```
consul snapshot save /path/to/snapshot; consul snapshot restore /path/to/snapshot
```


Best Practices to Harden CouchDB for DevSecOps
Disable admin party
Edit the CouchDB configuration file `local.ini` locat
`/opt/couchdb/etc/couchdb/`. Change the line `;
[admins]` to `[admins]`, and add your admin usern
and password. Restart CouchDB.
Example command: `sudo nano
/opt/couchdb/etc/couchdb/local.ini`
Restrict access to configuration files
Change the owner and group of
configuration directory `/opt/co
the CouchDB user and group.
Example command: `sudo chow
/opt/couchdb/etc/couchdb/`
Use SSL/TLS encryption
Create SSL/TLS certificates and configure Cou
use HTTPS.
Example command: `sudo openssl req -x509
days 365 -newkey rsa:2048 -keyout
/etc/ssl/private/couchdb.key -out
/etc/ssl/certs/couchdb.crt`
Limit access to ports
Use a firewall to limit access to only the necessary
Example command: `sudo ufw allow from 192.168
to any port 5984`
Update CouchDB regularly
Install updates and security patches regula
Example command: `sudo apt-get update &
upgrade`

CouchDB Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden CouchDB for DevSecOps

Disable admin party

Edit the CouchDB configuration file `local.ini` located at `/opt/couchdb/etc/couchdb/`. Change the line `; [admins] to [admins]`, and add your admin username and password. Save and exit the file. Restart CouchDB. Example command: `sudo nano /opt/couchdb/etc/couchdb/local.ini`

Restrict access to configuration files

Change the owner and group of the CouchDB configuration directory `/opt/couchdb/etc/couchdb/` to the CouchDB user and group. Example command: `sudo chown -R couchdb:couchdb /opt/couchdb/etc/couchdb/`

Use SSL/TLS encryption

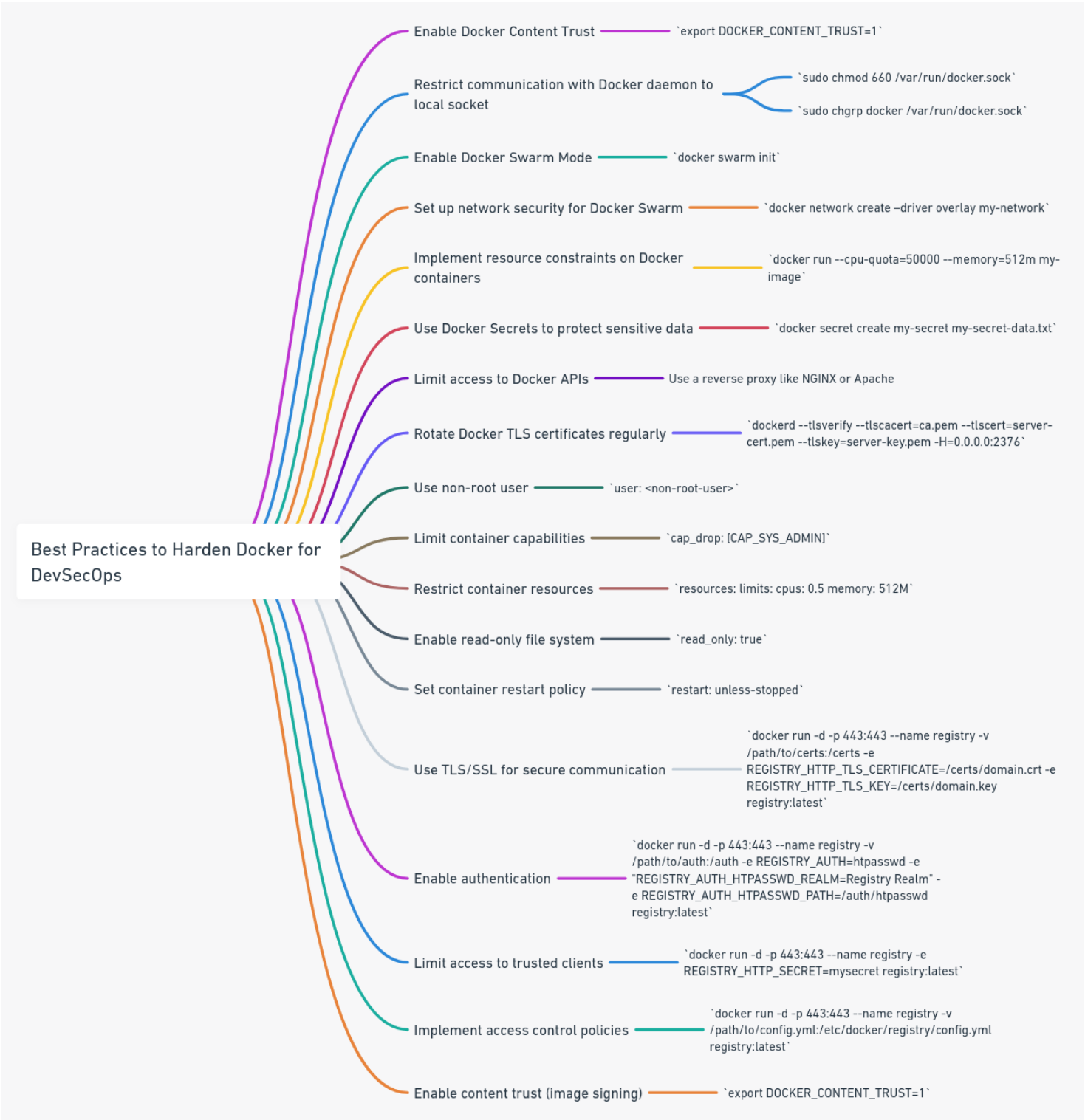
Create SSL/TLS certificates and configure CouchDB to use HTTPS. Example command for creating self-signed certificates: `sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/couchdb.key -out /etc/ssl/certs/couchdb.crt`

Limit access to ports

Use a firewall to limit access to only the necessary ports. Example command using `ufw`: `sudo ufw allow from 192.168.1.0/24 to any port 5984`

Update CouchDB regularly

Install updates and security patches regularly to keep the system secure. Example command for updating packages: `sudo apt-get update && sudo apt-get upgrade`


Best Practices to Harden Docker for DevSecOps
Enable Docker Content Trust
`export DOCKER_CONTENT_TRUST=1`
Restrict communication with Docker daemon to local socket
`sudo chmod 660 /var/run/docker.sock`
`sudo chgrp docker /var/run/docker.sock`
Enable Docker Swarm Mode
`docker swarm init`
Set up network security for Docker Swarm
`docker network create –driver overlay my-network`
Implement resource constraints on Docker containers
`docker run --cpu-quota=50000 --memory=512m my-image`
Use Docker Secrets to protect sensitive data
`docker secret create my-secret my-secret-data.txt`
Limit access to Docker APIs
Use a reverse proxy like NGINX or Apache
Rotate Docker TLS certificates regularly
`dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --tlskey=server-key.pem -H=0.0.0.0:2376`
Use non-root user
`user: <non-root-user>`
Limit container capabilities
`cap_drop: [CAP_SYS_ADMIN]`
Restrict container resources
`resources: limits: cpus: 0.5 memory: 512M`
Enable read-only file system
`read_only: true`
Set container restart policy
`restart: unless-stopped`
Use TLS/SSL for secure communication
`docker run -d -p 443:443 --name registry -v /path/to/certs:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key registry:latest`
Enable authentication
`docker run -d -p 443:443 --name registry -v /path/to/auth:/auth -e REGISTRY_AUTH=htpasswd -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd registry:latest`
Limit access to trusted clients
`docker run -d -p 443:443 --name registry -e REGISTRY_HTTP_SECRET=mysecret registry:latest`
Implement access control policies
`docker run -d -p 443:443 --name registry -v /path/to/config.yml:/etc/docker/registry/config.yml registry:latest`
Enable content trust (image signing)
`export DOCKER_CONTENT_TRUST=1`

Docker Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Docker for DevSecOps

Enable Docker Content Trust

```
export DOCKER_CONTENT_TRUST=1
```

Restrict communication with Docker daemon to local socket

sudo chmod 660 /var/run/docker.sock
sudo chgrp docker /var/run/docker.sock

Enable Docker Swarm Mode

docker swarm init

Set up network security for Docker Swarm

docker network create –driver overlay my-network

Implement resource constraints on Docker containers

```
docker run --cpu-quota=50000 --memory=512m my-image
```

Use Docker Secrets to protect sensitive data

```
docker secret create my-secret my-secret-data.txt
```

Limit access to Docker APIs

Use a reverse proxy like NGINX or Apache to limit access to the Docker API endpoint

Rotate Docker TLS certificates regularly

```
dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --tlskey=server-key.pem -H=0.0.0.0:2376
```

Use non-root user

```
user: <non-root-user>
```

Limit container capabilities

```
cap_drop: [CAP_SYS_ADMIN]
```

Restrict container resources

```
resources:
limits:
cpus: 0.5
memory: 512M
```

Enable read-only file system

```
read_only: true
```

Set container restart policy

```
restart: unless-stopped
```

Use TLS/SSL for secure communication

```
docker run -d -p 443:443 --name registry -v /path/to/certs:/certs -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt -e
REGISTRY_HTTP_TLS_KEY=/certs/domain.key registry:latest
```

Enable authentication

```
docker run -d -p 443:443 --name registry -v /path/to/auth:/auth -e REGISTRY_AUTH=htpasswd -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" -e
REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd registry:latest
```

Limit access to trusted clients

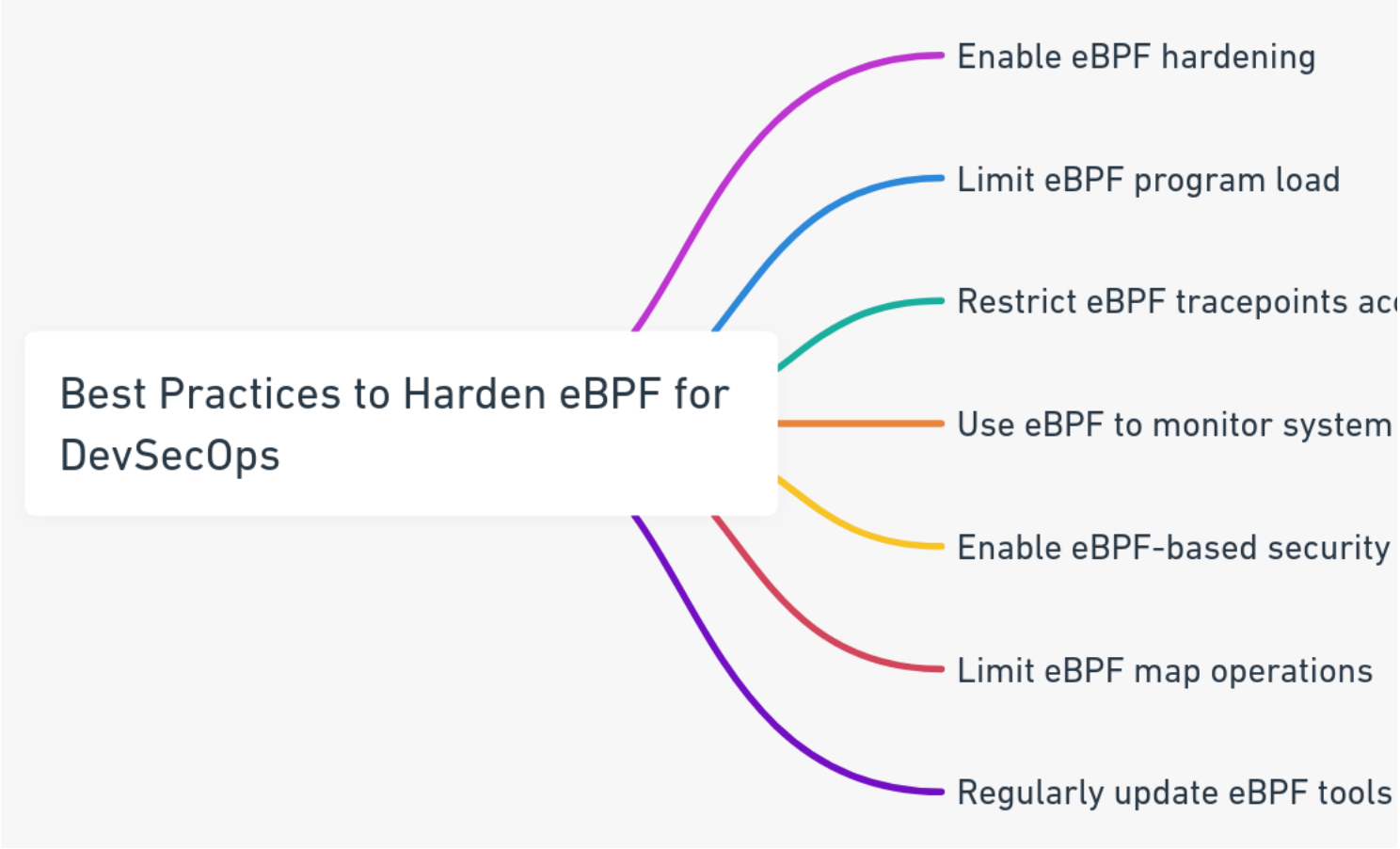
```
docker run -d -p 443:443 --name registry -e REGISTRY_HTTP_SECRET=mysecret registry:latest
```

Implement access control policies

```
docker run -d -p 443:443 --name registry -v /path/to/config.yml:/etc/docker/registry/config.yml registry:latest
```

Enable content trust (image signing)

```
export DOCKER_CONTENT_TRUST=1
```


Best Practices to Harden eBPF for DevSecOps
Enable eBPF hardening
Limit eBPF program load
Restrict eBPF tracepoints ac
Use eBPF to monitor system
Enable eBPF-based security
Limit eBPF map operations
Regularly update eBPF tools

eBPF Security Checklist for DevSecOps

TABLE OF CONTENTS

List of some best practices to eBPF for DevSecOps

Enable eBPF hardening

```
echo 1 > /proc/sys/net/core/bpf_jit_harden
```

Limit eBPF program load

```
setcap cap_bpf=e /path/to/program
```

Restrict eBPF tracepoints access

```
echo 0 > /proc/sys/kernel/perf_event_paranoid
```

Use eBPF to monitor system calls

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count(); }'
```

Enable eBPF-based security monitoring

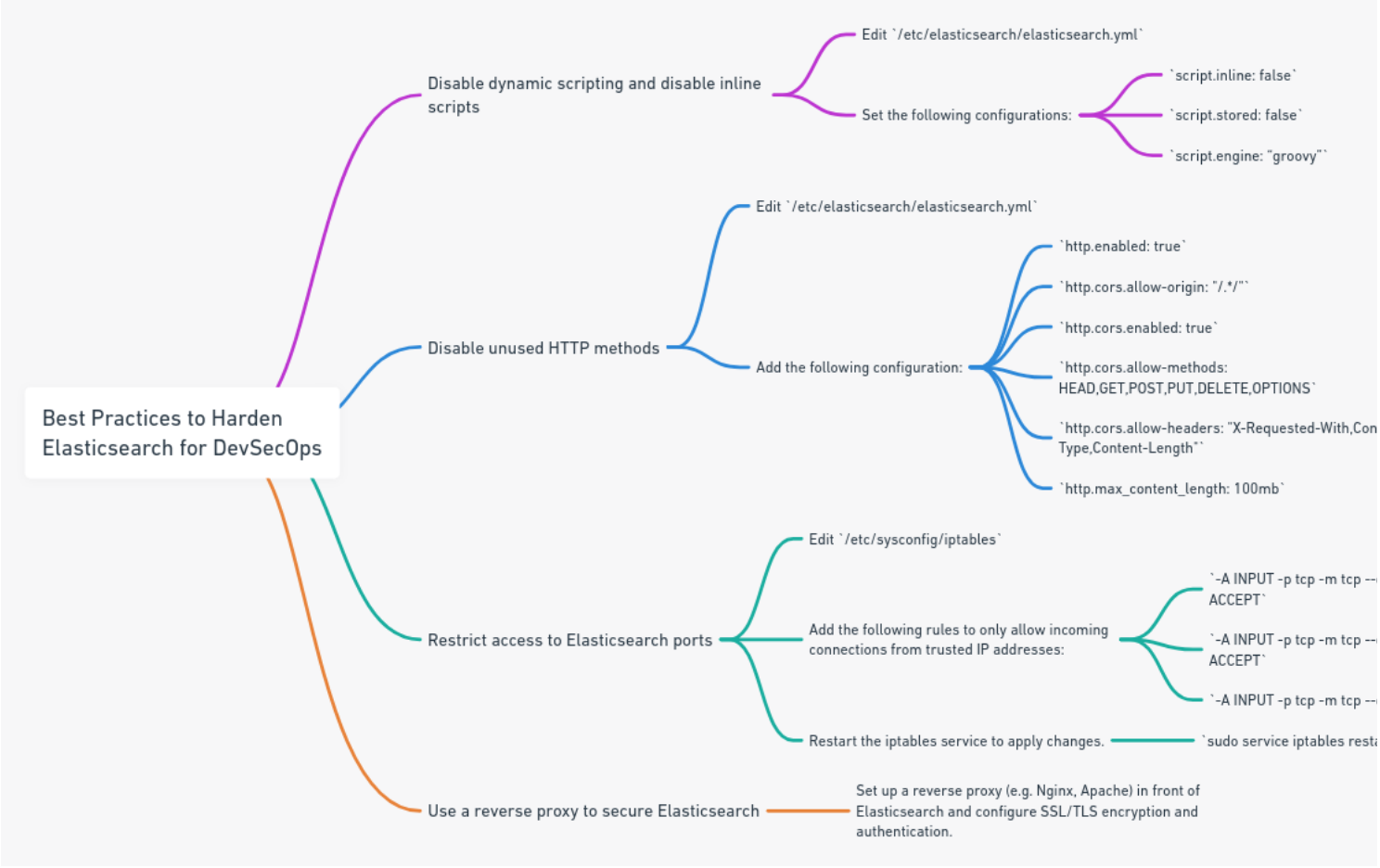
```
bpftool prog load secmon.bpf /sys/fs/bpf/
```

Limit eBPF map operations

```
bpftool map create /sys/fs/bpf/my_map type hash key 4 value 4 entries 1024
```

Regularly update eBPF tools and libraries

```
apt-get update && apt-get upgrade libbpf-tools
```


Best Practices to Harden Elasticsearch for DevSecOps
Disable dynamic scripting and disable inline scripts
Edit `/etc/elasticsearch/elasticsearch.yml`
Set the following configurations:
`script.inline: false`
`script.stored: false`
`script.engine: "groovy"`
Disable unused HTTP methods
Edit `/etc/elasticsearch/elasticsearch.yml`
Add the following configuration:
`http.enabled: true`
`http.cors.allow-origin: "/.*/"`
`http.cors.enabled: true`
`http.cors.allow-methods: HEAD,GET,POST,PUT,DELETE,OPTIONS`
`http.cors.allow-headers: "X-Requested-With,Con
Type,Content-Length"`
`http.max_content_length: 100mb`
Restrict access to Elasticsearch ports
Edit `/etc/sysconfig/iptables`
Add the following rules to only allow incoming connections from trusted IP addresses:
`-A INPUT -p tcp -m tcp --
ACCEPT`
`-A INPUT -p tcp -m tcp --
ACCEPT`
`-A INPUT -p tcp -m tcp --
Restart the iptables service to apply changes.
`sudo service iptables resta
Use a reverse proxy to secure Elasticsearch
Set up a reverse proxy (e.g. Nginx, Apache) in front of Elasticsearch and configure SSL/TLS encryption and authentication.

Elasticsearch Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Elasticsearch for DevSecOps

Disable dynamic scripting and disable inline scripts

sudo nano /etc/elasticsearch/elasticsearch.yml
Set the following configurations:
script.inline: false
script.stored: false
script.engine: “groovy”

Disable unused HTTP methods

`sudo nano /etc/elasticsearch/elasticsearch.yml` Add the following configuration:

```
http.enabled: true
```

`http.cors.allow-origin: "/.*/"` `http.cors.enabled: true`

```
http.cors.allow-methods: HEAD,GET,POST,PUT,DELETE,OPTIONS
```

```
http.cors.allow-headers: "X-Requested-With,Content-Type,Content-Length"
```

```
http.max_content_length: 100mb
```

Restrict access to Elasticsearch ports

```
sudo nano /etc/sysconfig/iptables
```

Add the following rules to only allow incoming connections from trusted IP addresses:

```
-A INPUT -p tcp -m tcp --dport 9200 -s 10.0.0.0/8 -j ACCEPT
```

```
-A INPUT -p tcp -m tcp --dport 9200 -s 192.168.0.0/16 -j ACCEPT
```

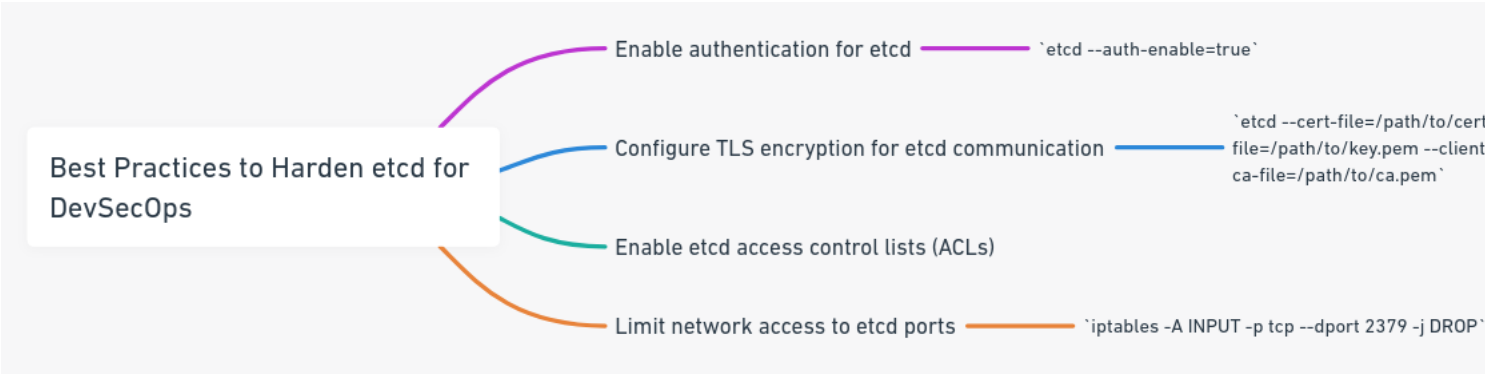
```
-A INPUT -p tcp -m tcp --dport 9200 -j DROP
```

Restart the iptables service to apply changes.

```
sudo service iptables restart
```

Use a reverse proxy to secure Elasticsearch

Set up a reverse proxy (e.g. Nginx, Apache) in front of Elasticsearch and configure SSL/TLS encryption and authentication.


Best Practices to Harden etcd for DevSecOps
Enable authentication for etcd
`etcd --auth-enable=true`
Configure TLS encryption for etcd communication
`etcd --cert-file=/path/to/cert
file=/path/to/key.pem --client
ca-file=/path/to/ca.pem`
Enable etcd access control lists (ACLs)
Limit network access to etcd ports
`iptables -A INPUT -p tcp --dport 2379 -j DROP`

etcd Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden etcd for DevSecOps

Enable authentication for etcd

```
etcd --auth-enable=true
```

Configure TLS encryption for etcd communication

```
etcd --cert-file=/path/to/cert.pem --key-file=/path/to/key.pem --client-cert-auth=true --trusted-ca-file=/path/to/ca.pem
```

Enable etcd access control lists (ACLs)

```
Enable etcd access control lists (ACLs)
```

Limit network access to etcd ports

```
iptables -A INPUT -p tcp --dport 2379 -j DROP
```


Best Practices to Harden Git for DevSecOps
Enable GPG signature verification
`git config --global commit.gpgsign true`
Set a strong passphrase for GPG key
`gpg –edit-key` and then use the `passw
Use HTTPS instead of SSH for remote repositories
`git config --global url."https://".instead
Enable two-factor authentication
Enable it through the Git service provider's w
Set Git to ignore file mode changes
`git config --global core.fileMode false`
Configure Git to use a credential helper
`git config --global credential.helper <
Use signed commits
`git commit -S`
or
`git config --global commit.gpgsign true`
Set Git to automatically prune stale remote-tracking branches
`git config --global fetch.prune t
Set Git to always rebase instead of merge when pulling
`git config --global pull.reb
Use Git's ignore feature to exclude sensitive files
Add files or file patterns to

Git Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Git for DevSecOps

Enable GPG signature verification

```
git config --global commit.gpgsign true
```

Set a strong passphrase for GPG key

gpg –edit-key and then use the passwd command to set a strong passphrase

Use HTTPS instead of SSH for remote repositories

```
git config --global url."https://".insteadOf git://
```

Enable two-factor authentication

Enable it through the Git service provider's website

Set Git to ignore file mode changes

```
git config --global core.fileMode false
```

Configure Git to use a credential helper

`git config --global credential.helper <helper>` where `<helper>` is the name of the credential helper (e.g., `manager`, `store`)

Use signed commits

```
git commit -S
```

or

```
git config --global commit.gpgsign true
```

Set Git to automatically prune stale remote-tracking branches

```
git config --global fetch.prune true
```

Set Git to always rebase instead of merge when pulling

```
git config --global pull.rebase true
```

Use Git's `ignore` feature to exclude sensitive files

Add files or file patterns to the `.gitignore` file


Best Practices to Harden GitLab for DevSecOps
Update GitLab to the latest version
`sudo apt-get update && sudo apt-get upgrade gitlab-ee`
Enable SSL/TLS for GitLab
Edit `/etc/gitlab/gitlab.rb` and add the necessary configurations
Run `sudo gitlab-ctl reconfigure`
Disable GitLab sign up
Edit `/etc/gitlab/gitlab.rb` and add `gitlab_rails['gitlab_signup_enabled'] = false`
Run `sudo gitlab-ctl reconfigure`
Set a strong password policy
Edit `/etc/gitlab/gitlab.rb` and add the necessary configurations
Run `sudo gitlab-ctl reconfigure`
Limit the maximum file size
Edit `/etc/gitlab/gitlab.rb` and add `gitlab_rails['max_attachment_size'] = 10.megabytes`
Run `sudo gitlab-ctl reconfigure`
Enable two-factor authentication (2FA)
Go to GitLab's web interface and follow the prompts to set up 2FA
Enable audit logging
Edit `/etc/gitlab/gitlab.rb` and add `gitlab_rails['audit_events_enabled'] = true`
Run `sudo gitlab-ctl reconfigure`
Configure GitLab backups
Edit `/etc/gitlab/gitlab.rb` and add the necessary configurations
Run `sudo gitlab-ctl reconfigure`
Restrict SSH access
Edit `/etc/gitlab/gitlab.rb` and add `gitlab_rails['gitlab_shell_ssh_port'] = 22`
Run `sudo gitlab-ctl reconfigure`
Enable firewall rules
Configure your firewall to only allow incoming traffic on necessary ports

Gitlab Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Gitlab for DevSecOps

Update GitLab to the latest version

```
sudo apt-get update && sudo apt-get upgrade gitlab-ee
```

Enable SSL/TLS for GitLab

Edit /etc/gitlab/gitlab.rb and add the following lines:
external_url 'https://gitlab.example.com'
nginx['redirect_http_to_https'] = true
nginx['ssl_certificate'] = "/etc/gitlab/ssl/gitlab.example.com.crt"
nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/gitlab.example.com.key"
gitlab_rails['gitlab_https'] = true
gitlab_rails['trusted_proxies'] = ['192.168.1.1'] (replace 192.168.1.1 with the IP address of your proxy)
Then run sudo gitlab-ctl reconfigure

Disable GitLab sign up

Edit /etc/gitlab/gitlab.rb and add the following line:
gitlab_rails['gitlab_signup_enabled'] = false
Then run sudo gitlab-ctl reconfigure

Set a strong password policy

Edit /etc/gitlab/gitlab.rb and add the following lines:
gitlab_rails['password_minimum_length'] = 12
gitlab_rails['password_complexity'] = 2
Then run sudo gitlab-ctl reconfigure

Limit the maximum file size

Edit /etc/gitlab/gitlab.rb and add the following line:
gitlab_rails['max_attachment_size'] = 10.megabytes

Then run sudo gitlab-ctl reconfigure

Enable two-factor authentication (2FA)

Go to GitLab's web interface, click on your profile picture in the top-right corner, and select "Settings". Then select "Account" from the left-hand menu and follow the prompts to set up 2FA.

Enable audit logging

Edit /etc/gitlab/gitlab.rb and add the following line:
gitlab_rails['audit_events_enabled'] = true
Then run sudo gitlab-ctl reconfigure

Configure GitLab backups

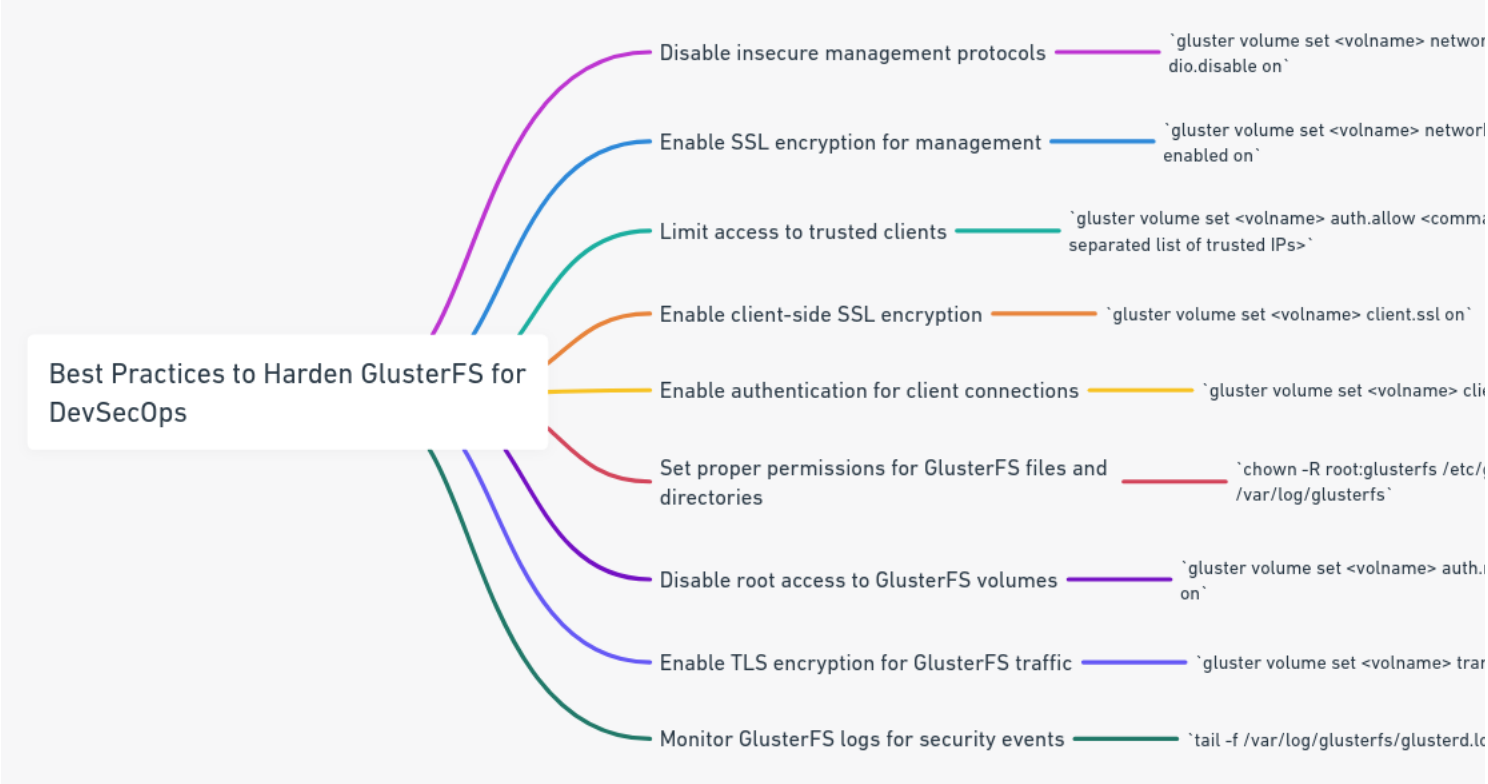
Edit /etc/gitlab/gitlab.rb and add the following lines:
gitlab_rails['backup_keep_time'] = 604800
gitlab_rails['backup_archive_permissions'] = 0644
gitlab_rails['backup_pg_schema'] = 'public'
gitlab_rails['backup_path'] = "/var/opt/gitlab/backups"
Then run sudo gitlab-ctl reconfigure

Restrict SSH access

Edit /etc/gitlab/gitlab.rb and add the following line:
gitlab_rails['gitlab_shell_ssh_port'] = 22
Then run sudo gitlab-ctl reconfigure

Enable firewall rules

Configure your firewall to only allow incoming traffic on ports that are necessary for GitLab to function, such as 80, 443, and 22. Consult your firewall documentation for instructions on how to configure the firewall rules.


Best Practices to Harden GlusterFS for DevSecOps
Disable insecure management protocols
`gluster volume set <volname> networ
dio.disable on`
Enable SSL encryption for management
`gluster volume set <volname> networ
enabled on`
Limit access to trusted clients
`gluster volume set <volname> auth.allow <comma
separated list of trusted IPs>`
Enable client-side SSL encryption
`gluster volume set <volname> client.ssl on`
Enable authentication for client connections
`gluster volume set <volname> cli
Set proper permissions for GlusterFS files and directories
`chown -R root:glusterfs /etc/
/var/log/glusterfs`
Disable root access to GlusterFS volumes
`gluster volume set <volname> auth.
on`
Enable TLS encryption for GlusterFS traffic
`gluster volume set <volname> tra
Monitor GlusterFS logs for security events
`tail -f /var/log/glusterfs/glusterd.lo

GlusterFS Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden GlusterFS for DevSecOps

Disable insecure management protocols

```
gluster volume set <volname> network.remote-dio.disable on
```

Enable SSL encryption for management

```
gluster volume set <volname> network.remote.ssl-enabled on
```

Limit access to trusted clients

```
gluster volume set <volname> auth.allow <comma-separated list of trusted IPs>
```

Enable client-side SSL encryption

```
gluster volume set <volname> client.ssl on
```

Enable authentication for client connections

```
gluster volume set <volname> client.auth on
```

Set proper permissions for GlusterFS files and directories

```
chown -R root:glusterfs /etc/glusterfs /var/lib/glusterd /var/log/glusterfs
```

Disable root access to GlusterFS volumes

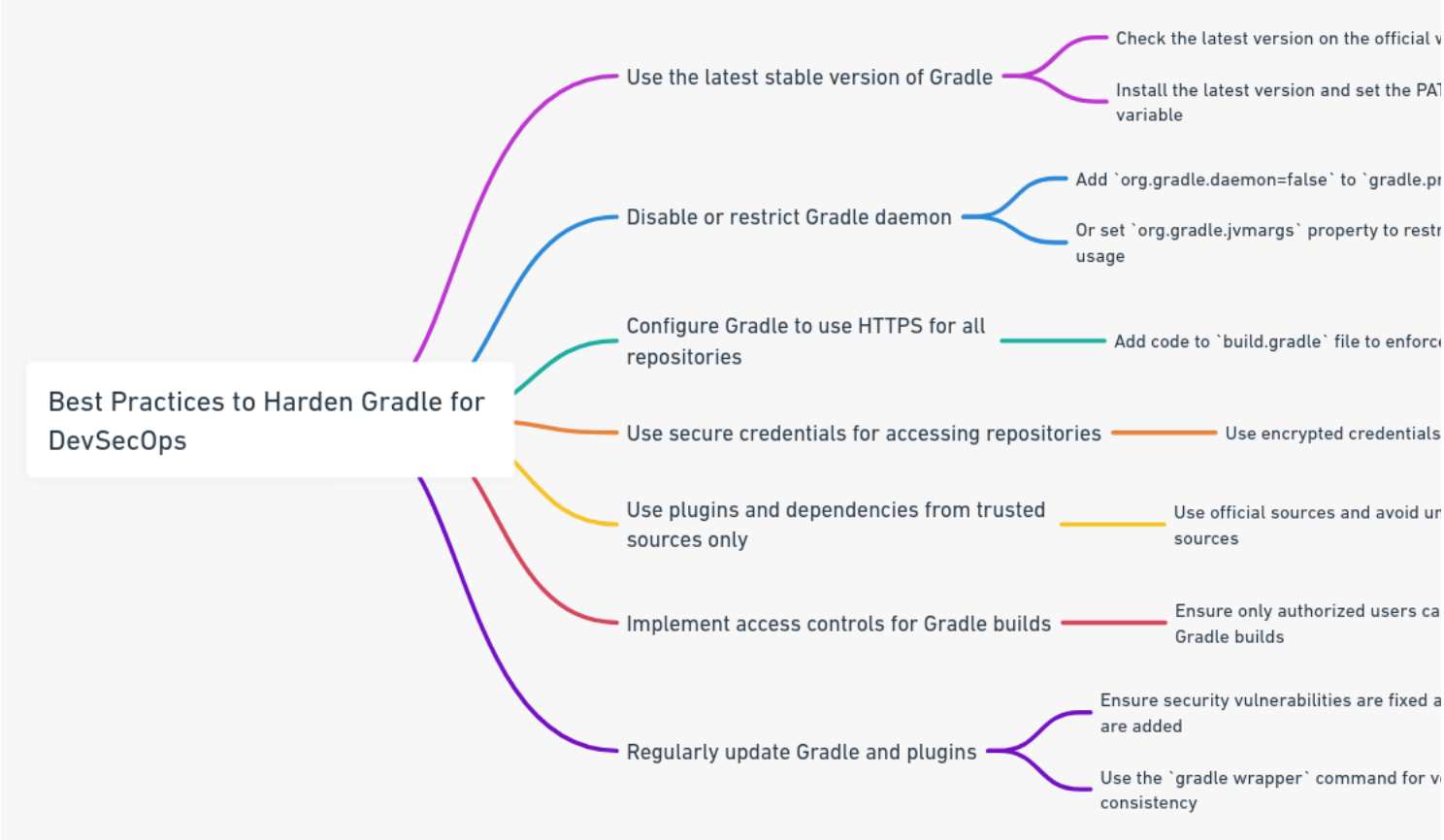
```
gluster volume set <volname> auth.reject-unauthorized on
```

Enable TLS encryption for GlusterFS traffic

```
gluster volume set <volname> transport-type
```

Monitor GlusterFS logs for security events

```
tail -f /var/log/glusterfs/glusterd.log
```


Best Practices to Harden Gradle for DevSecOps
Use the latest stable version of Gradle
Check the latest version on the official v
Install the latest version and set the PAT variable
Disable or restrict Gradle daemon
Add `org.gradle.daemon=false` to `gradle.pr
Or set `org.gradle.jvmargs` property to restr usage
Configure Gradle to use HTTPS for all repositories
Add code to `build.gradle` file to enforc
Use secure credentials for accessing repositories
Use encrypted credentials
Use plugins and dependencies from trusted sources only
Use official sources and avoid ur sources
Implement access controls for Gradle builds
Ensure only authorized users ca Gradle builds
Regularly update Gradle and plugins
Ensure security vulnerabilities are fixed a are added
Use the `gradle wrapper` command for v consistency

Gradle Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Gradle for DevSecOps

Use the latest stable version of Gradle

Check the latest version on the official website: https://gradle.org/releases/, and then install it. For example: wget https://services.gradle.org/distributions/gradle-7.0.2-bin.zip, unzip gradle-7.0.2-bin.zip, and set the PATH environment variable to the Gradle bin directory.

Disable or restrict Gradle daemon

You can disable the daemon by adding the following line to the gradle.properties file: org.gradle.daemon=false. Alternatively, you can restrict the maximum amount of memory that can be used by the daemon by setting the org.gradle.jvmargs property.

Configure Gradle to use HTTPS for all repositories

Add the following code to the build.gradle file to enforce using HTTPS for all repositories:

```
allprojects {
    repositories {
        mavenCentral {
            url "https://repo1.maven.org/maven2/"
        }
        maven {
            url "https://plugins.gradle.org/m2/"
        }
    }
}
```

Use secure credentials for accessing repositories

Use encrypted credentials in the `build.gradle` file or environment variables for accessing repositories.

Use plugins and dependencies from trusted sources only

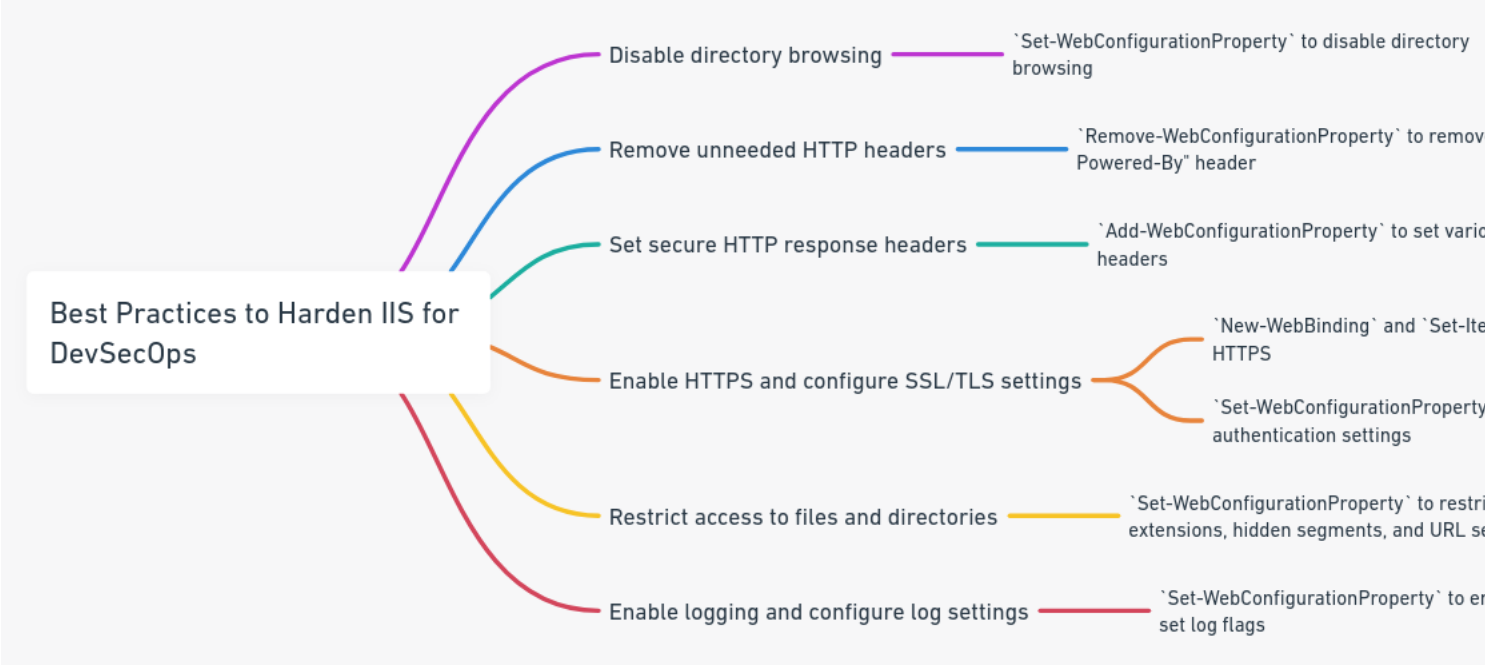
Use plugins and dependencies from official sources, and avoid using those from unknown or untrusted sources.

Implement access controls for Gradle builds

Implement access controls to ensure that only authorized users can execute or modify Gradle builds.

Regularly update Gradle and plugins

Regularly update Gradle and its plugins to ensure that security vulnerabilities are fixed and new features are added. Use the `gradle wrapper` command to ensure that all team members use the same version of Gradle.


Best Practices to Harden IIS for DevSecOps
Disable directory browsing
`Set-WebConfigurationProperty` to disable directory browsing
Remove unneeded HTTP headers
`Remove-WebConfigurationProperty` to remov
Powered-By" header
Set secure HTTP response headers
`Add-WebConfigurationProperty` to set varic
headers
Enable HTTPS and configure SSL/TLS settings
`New-WebBinding` and `Set-Ite
HTTPS
`Set-WebConfigurationProperty
authentication settings
Restrict access to files and directories
`Set-WebConfigurationProperty` to restri
extensions, hidden segments, and URL se
Enable logging and configure log settings
`Set-WebConfigurationProperty` to er
set log flags

IIS Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden IIS for DevSecOps

Disable directory browsing

```
Set-WebConfigurationProperty -filter /system.webServer/directoryBrowse -PSPath "IIS:\Sites\Default Web Site" -name enabled -value $false
```

Remove unneeded HTTP headers

```
Remove-WebConfigurationProperty -filter "system.webServer/httpProtocol/customHeaders" -name ."X-Powered-By"
```

Set secure HTTP response headers

```
Add-WebConfigurationProperty -filter "system.webServer/staticContent" -name "clientCache.cacheControlMode" -value "UseMaxAge"<br>Set-
WebConfigurationProperty -filter "system.webServer/staticContent/clientCache" -name "cacheControlMaxAge" -value "365.00:00:00"<br>Add-
WebConfigurationProperty -filter "system.webServer/httpProtocol/customHeaders" -name "X-Content-Type-Options" -value "nosniff"<br>Add-
WebConfigurationProperty -filter "system.webServer/httpProtocol/customHeaders" -name "X-Frame-Options" -value "SAMEORIGIN"<br>Add-
WebConfigurationProperty -filter "system.webServer/httpProtocol/customHeaders" -name "X-XSS-Protection" -value "1; mode=block"
```

Enable HTTPS and configure SSL/TLS settings

```
New-WebBinding -Name "Default Web Site" -Protocol https -Port 443 -IPAddress "*" -SslFlags 1<br>Set-ItemProperty -Path
IIS:\SslBindings\0.0.0.0!443 -Name "SslFlags" -Value "1"<br>Set-WebConfigurationProperty -filter
"system.webServer/security/authentication/iisClientCertificateMappingAuthentication" -name enabled -value $false<br>Set-WebConfigurationProperty -
filter "system.webServer/security/authentication/anonymousAuthentication" -name enabled -value $false<br>Set-WebConfigurationProperty -filter
"system.webServer/security/authentication/basicAuthentication" -name enabled -value $false<br>Set-WebConfigurationProperty -filter
"system.webServer/security/authentication/digestAuthentication" -name enabled -value $false<br>Set-WebConfigurationProperty -filter
"system.webServer/security/authentication/windowsAuthentication" -name enabled -value $true<br>Set-WebConfigurationProperty -filter
"system.webServer/security/authentication/windowsAuthentication" -name useKernelMode -value $true
```

Restrict access to files and directories

```
Set-WebConfigurationProperty -filter "/system.webServer/security/requestFiltering/fileExtensions" -name "." -value @{allowed="$false"}<br>Set-
WebConfigurationProperty -filter "/system.webServer/security/requestFiltering/hiddenSegments" -name "." -value @{allowed="$false"}<br>Set-
```

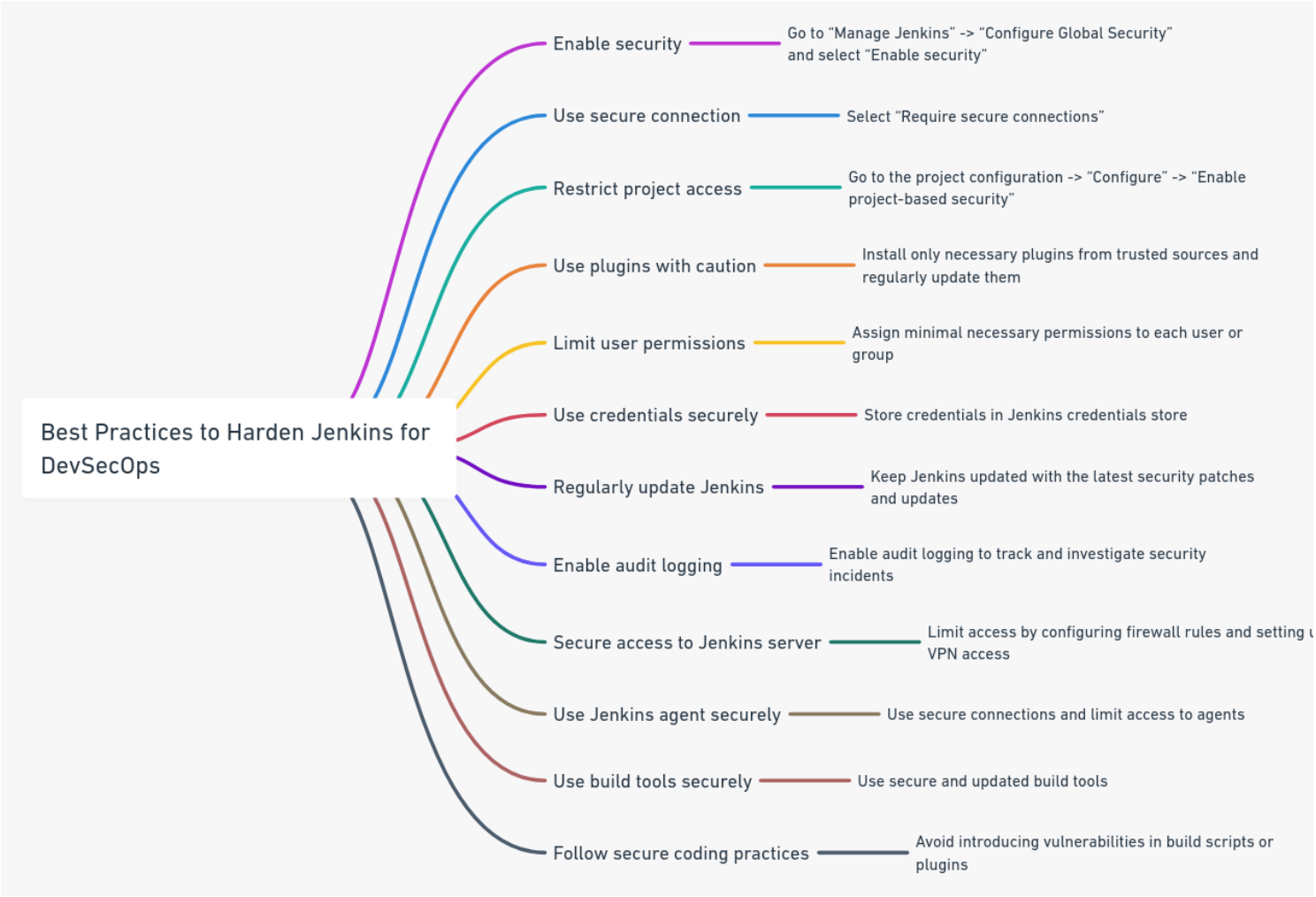
```
WebConfigurationProperty -filter "/system.webServer/security/requestFiltering/denyUrlSequences" -name "." -value @{add="$false"}
```

Enable logging and configure log settings

```
Set-WebConfigurationProperty -filter "/system.webServer/httpLogging" -name dontLog -value $false
```

or

```
Set-WebConfigurationProperty -filter "/system.webServer/httpLogging" -name logExtFileFlags -value "Date, Time, ClientIP, UserName, SiteName,
ComputerName, ServerIP, Method, UriStem, UriQuery, HttpStatus, Win32Status, BytesSent, BytesRecv, TimeTaken
```


Best Practices to Harden Jenkins for DevSecOps
Enable security
Go to “Manage Jenkins” -> “Configure Global Security” and select “Enable security”
Use secure connection
Select “Require secure connections”
Restrict project access
Go to the project configuration -> “Configure” -> “Enable project-based security”
Use plugins with caution
Install only necessary plugins from trusted sources and regularly update them
Limit user permissions
Assign minimal necessary permissions to each user or group
Use credentials securely
Store credentials in Jenkins credentials store
Regularly update Jenkins
Keep Jenkins updated with the latest security patches and updates
Enable audit logging
Enable audit logging to track and investigate security incidents
Secure access to Jenkins server
Limit access by configuring firewall rules and setting u
VPN access
Use Jenkins agent securely
Use secure connections and limit access to agents
Use build tools securely
Use secure and updated build tools
Follow secure coding practices
Avoid introducing vulnerabilities in build scripts or plugins

Jenkins Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Jenkins for DevSecOps

Enable security

Go to “Manage Jenkins” -> “Configure Global Security” and select “Enable security”

Use secure connection

Go to “Manage Jenkins” -> “Configure Global Security” and select “Require secure connections”

Restrict project access

Go to the project configuration -> “Configure” -> “Enable project-based security”

Use plugins with caution

Install only necessary plugins from trusted sources and regularly update them

Limit user permissions

Assign minimal necessary permissions to each user or group

Use credentials securely

Store credentials in Jenkins credentials store and use them only where necessary

Regularly update Jenkins

Keep Jenkins updated with the latest security patches and updates

Enable audit logging

Enable audit logging to track and investigate security incidents

Secure access to Jenkins server

Limit access to Jenkins server by configuring firewall rules and setting up VPN access

Use Jenkins agent securely

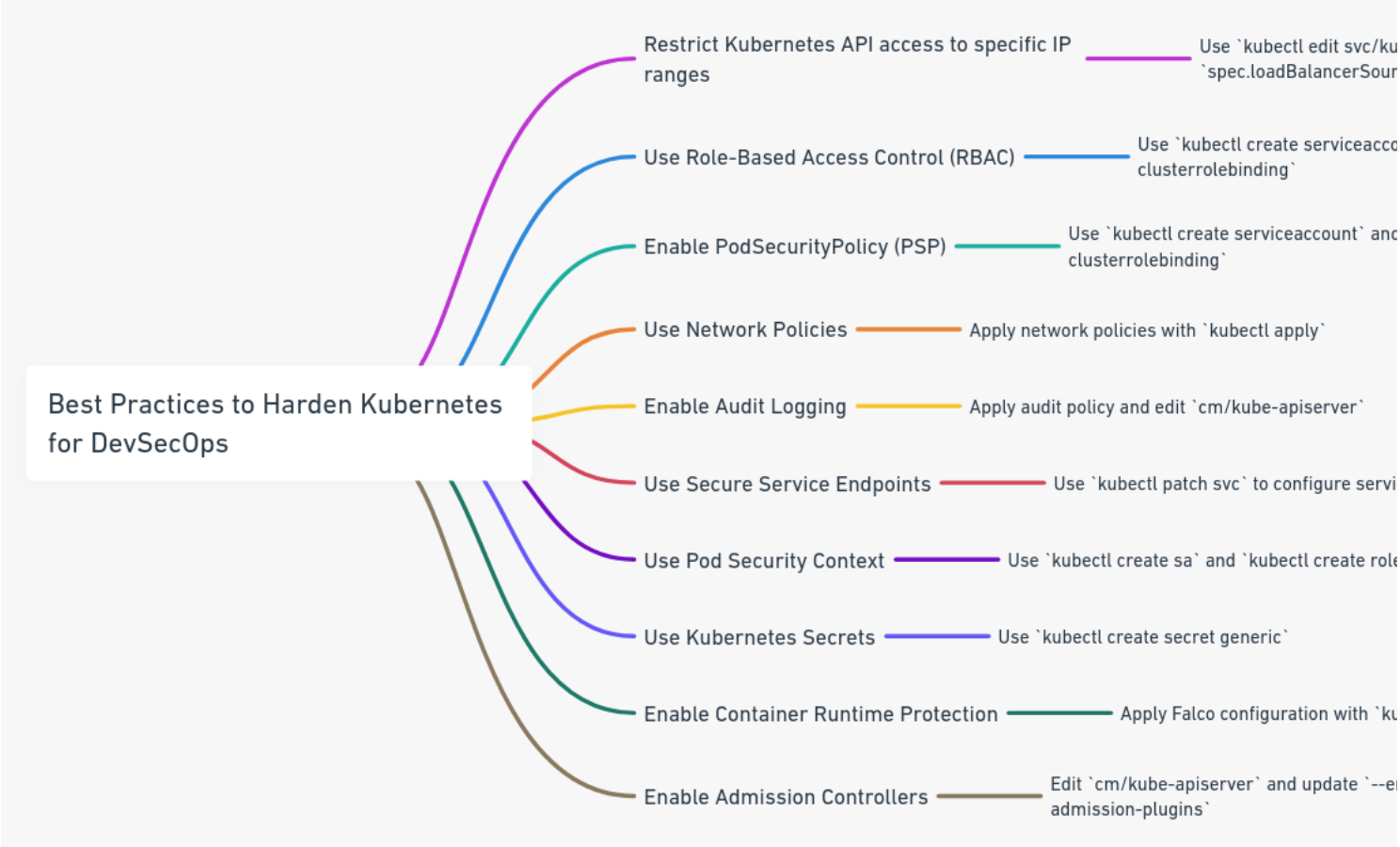
Use secure connections between Jenkins master and agents and limit access to agents

Use build tools securely

Use secure and updated build tools and avoid using system tools or commands directly in build scripts

Follow secure coding practices

Follow secure coding practices to avoid introducing vulnerabilities in build scripts or plugins


Best Practices to Harden Kubernetes for DevSecOps
Restrict Kubernetes API access to specific IP ranges
Use `kubectl edit svc/ku
`spec.loadBalancerSour
Use Role-Based Access Control (RBAC)
Use `kubectl create serviceacc
clusterrolebinding`
Enable PodSecurityPolicy (PSP)
Use `kubectl create serviceaccount` and
clusterrolebinding`
Use Network Policies
Apply network policies with `kubectl apply`
Enable Audit Logging
Apply audit policy and edit `cm/kube-apiserver`
Use Secure Service Endpoints
Use `kubectl patch svc` to configure servi
Use Pod Security Context
Use `kubectl create sa` and `kubectl create role
Use Kubernetes Secrets
Use `kubectl create secret generic`
Enable Container Runtime Protection
Apply Falco configuration with `ku
Enable Admission Controllers
Edit `cm/kube-apiserver` and update `--e
admission-plugins`

Kuberneties Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Kuberneties for DevSecOps

Restrict Kubernetes API access to specific IP ranges

`kubectl edit svc/kubernetes`

Update `spec.loadBalancerSourceRanges`

Use Role-Based Access Control (RBAC)

```
kubectl create serviceaccount <name> <br> kubectl create clusterrolebinding <name> --clusterrole=<role> --serviceaccount=<namespace>:<name>
```

Enable PodSecurityPolicy (PSP)

```
kubectl create serviceaccount psp-sa <br> kubectl create clusterrolebinding psp-binding --clusterrole=psp:vmxnet3 --serviceaccount=default:psp-sa
```

Use Network Policies

```
kubectl apply -f networkpolicy.yml
```

Enable Audit Logging

```
kubectl apply -f audit-policy.yaml <br> kubectl edit cm/kube-apiserver -n kube-system <br> Update --audit-log-path and --audit-policy-file
```

Use Secure Service Endpoints

```
kubectl patch svc <svc-name> -p '{"spec": {"publishNotReadyAddresses": true, "sessionAffinity": "ClientIP"}}'
```

Use Pod Security Context

`kubectl create sa pod-sa`

`kubectl create rolebinding pod-sa --role=psp:vmxnet3 --serviceaccount=default:pod-sa`

Use Kubernetes Secrets

```
kubectl create secret generic <name> --from-file=<path-to-file>
```

Enable Container Runtime Protection

```
kubectl apply -f falco.yaml
```

Enable Admission Controllers

`kubectl edit cm/kube-apiserver -n kube-system`

Update `--enable-admission-plugins`

Memcached Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Memcached for DevSecOps

Disable UDP listener

```
sed -i 's/^-U 0/#-U 0/g' /etc/sysconfig/memcached
```

Enable SASL authentication

`sed -i 's/^#-S/-S/g' /etc/sysconfig/memcached`

`yum install cyrus-sasl-plain`

`htpasswd -c /etc/sasl2/memcached-sasldb username`

`chmod 600 /etc/sasl2/memcached-sasldb`

Limit incoming traffic to known IP addresses

```
iptables -A INPUT -p tcp --dport 11211 -s 192.168.1.100 -j ACCEPT
```

Limit maximum memory usage

```
echo 'CACHESIZE="128"' > /etc/sysconfig/memcached
```

Run as non-root user

```
sed -i 's/^-u root/-u memcached/g' /etc/sysconfig/memcached
```

Enable logging

`sed -i 's/^logfile/#logfile/g' /etc/sysconfig/memcached`

`mkdir /var/log/memcached`

`touch /var/log/memcached/memcached.log`

`chown memcached:memcached /var/log/memcached/memcached.log`

`sed -i 's/^#logfile/LOGFILE="\/var\/log\/memcached\/memcached.log"/g' /etc/sysconfig/memcached`

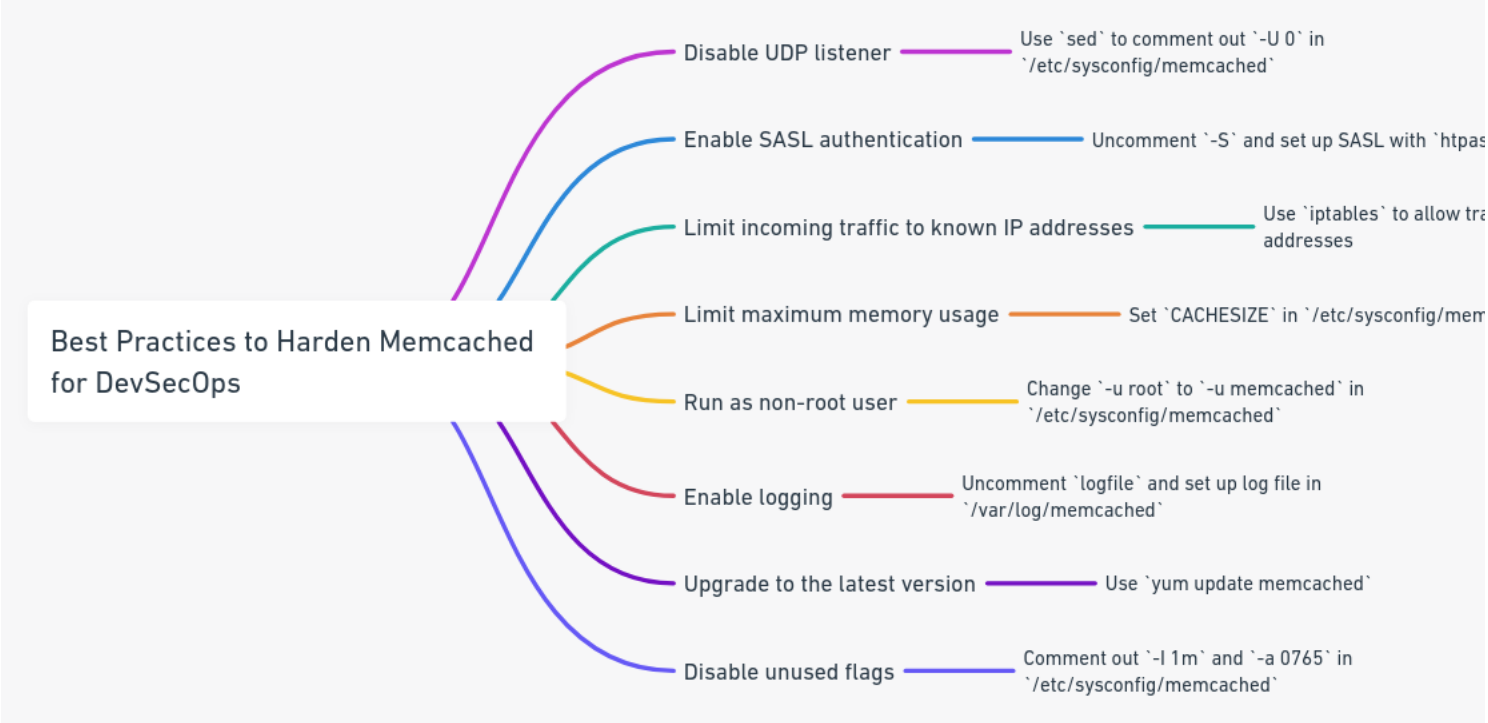
Upgrade to the latest version

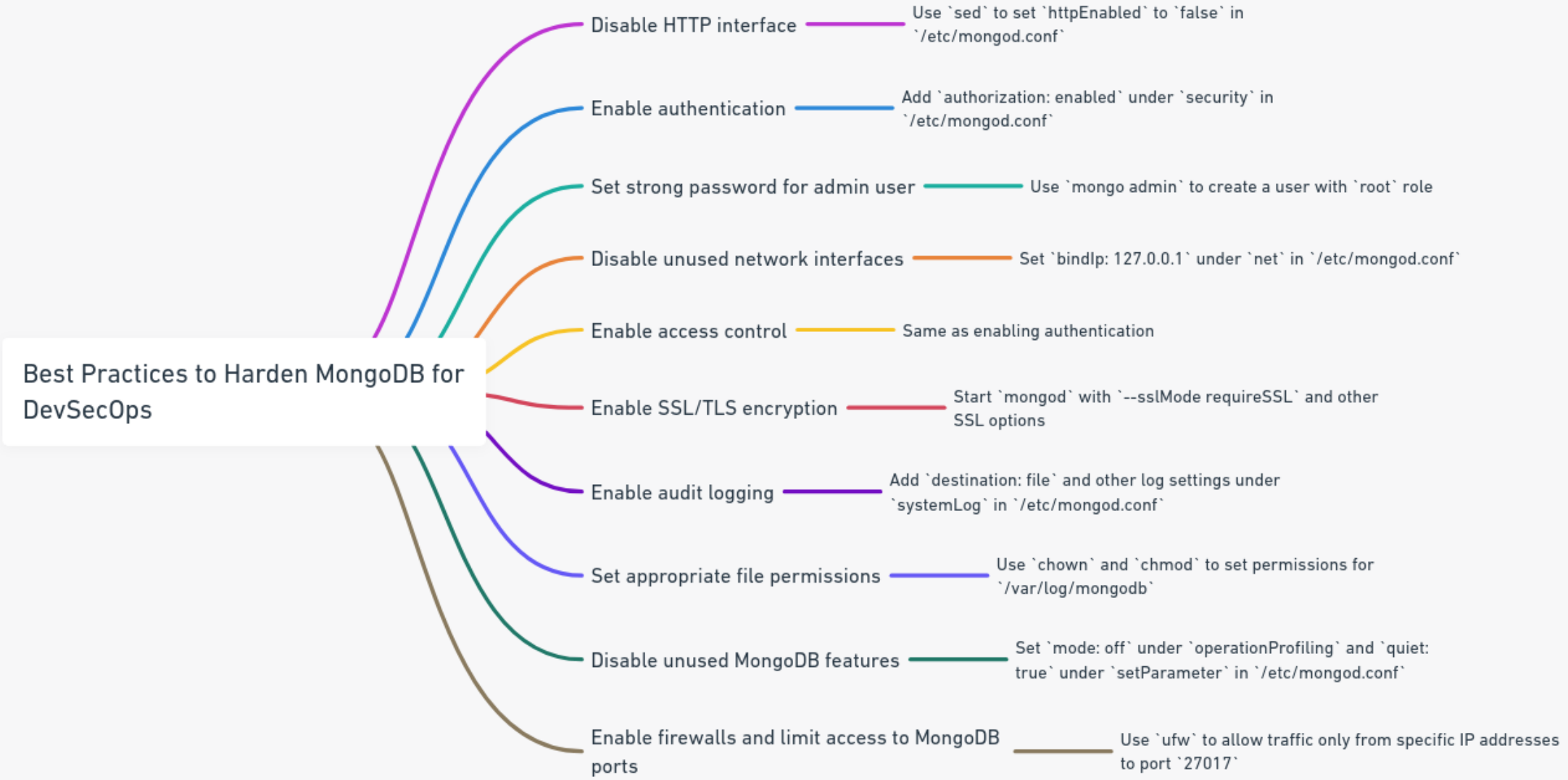
```
yum update memcached
```

Disable unused flags

`sed -i 's/^-I 1m/#-I 1m/g' /etc/sysconfig/memcached`

`sed -i 's/^-a 0765/#-a 0765/g' /etc/sysconfig/memcached`


Best Practices to Harden Memcached for DevSecOps
Disable UDP listener
Use `sed` to comment out `-U 0` in `/etc/sysconfig/memcached`
Enable SASL authentication
Uncomment `-S` and set up SASL with `htpas
Limit incoming traffic to known IP addresses
Use `iptables` to allow tra addresses
Limit maximum memory usage
Set `CACHESIZE` in `/etc/sysconfig/mem
Run as non-root user
Change `-u root` to `-u memcached` in `/etc/sysconfig/memcached`
Enable logging
Uncomment `logfile` and set up log file in `/var/log/memcached`
Upgrade to the latest version
Use `yum update memcached`
Disable unused flags
Comment out `-l 1m` and `-a 0765` in `/etc/sysconfig/memcached`


Best Practices to Harden MongoDB for DevSecOps
Disable HTTP interface
Use `sed` to set `httpEnabled` to `false` in `/etc/mongod.conf`
Enable authentication
Add `authorization: enabled` under `security` in `/etc/mongod.conf`
Set strong password for admin user
Use `mongo admin` to create a user with `root` role
Disable unused network interfaces
Set `bindIp: 127.0.0.1` under `net` in `/etc/mongod.conf`
Enable access control
Same as enabling authentication
Enable SSL/TLS encryption
Start `mongod` with `--sslMode requireSSL` and other SSL options
Enable audit logging
Add `destination: file` and other log settings under `systemLog` in `/etc/mongod.conf`
Set appropriate file permissions
Use `chown` and `chmod` to set permissions for `/var/log/mongodb`
Disable unused MongoDB features
Set `mode: off` under `operationProfiling` and `quiet: true` under `setParameter` in `/etc/mongod.conf`
Enable firewalls and limit access to MongoDB ports
Use `ufw` to allow traffic only from specific IP addresses to port `27017`

MongoDB Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden MongoDB for DevSecOps

Disable HTTP interface

```
sed -i '/httpEnabled/ s/true/false/g' /etc/mongod.conf
```

Enable authentication

```
sed -i '/security:/a \ \ \ \ authorization: enabled' /etc/mongod.conf
```

Set strong password for admin user

```
mongo admin --eval "db.createUser({user: 'admin', pwd: 'new_password_here', roles: ['root']})"
```

Disable unused network interfaces

```
sed -i '/net:/a \ \ \ \ bindIp: 127.0.0.1' /etc/mongod.conf
```

Enable access control

```
sed -i '/security:/a \ \ \ \ authorization: enabled' /etc/mongod.conf
```

Enable SSL/TLS encryption

```
mongod --sslMode requireSSL --sslPEMKeyFile /path/to/ssl/key.pem --sslCAFile /path/to/ca/ca.pem --sslAllowInvalidHostnames
```

Enable audit logging

```
sed -i '/systemLog:/a \ \ \ \ destination: file\n\ \ \ \ path: /var/log/mongodb/audit.log\n\ \ \ \ logAppend: true\n\ \ \ \ auditLog:\n\ \ \ \ \ \ \ \ destination: file\n\ \ \ \ \ \ \ \ format: JSON' /etc/mongod.conf
```

Set appropriate file permissions

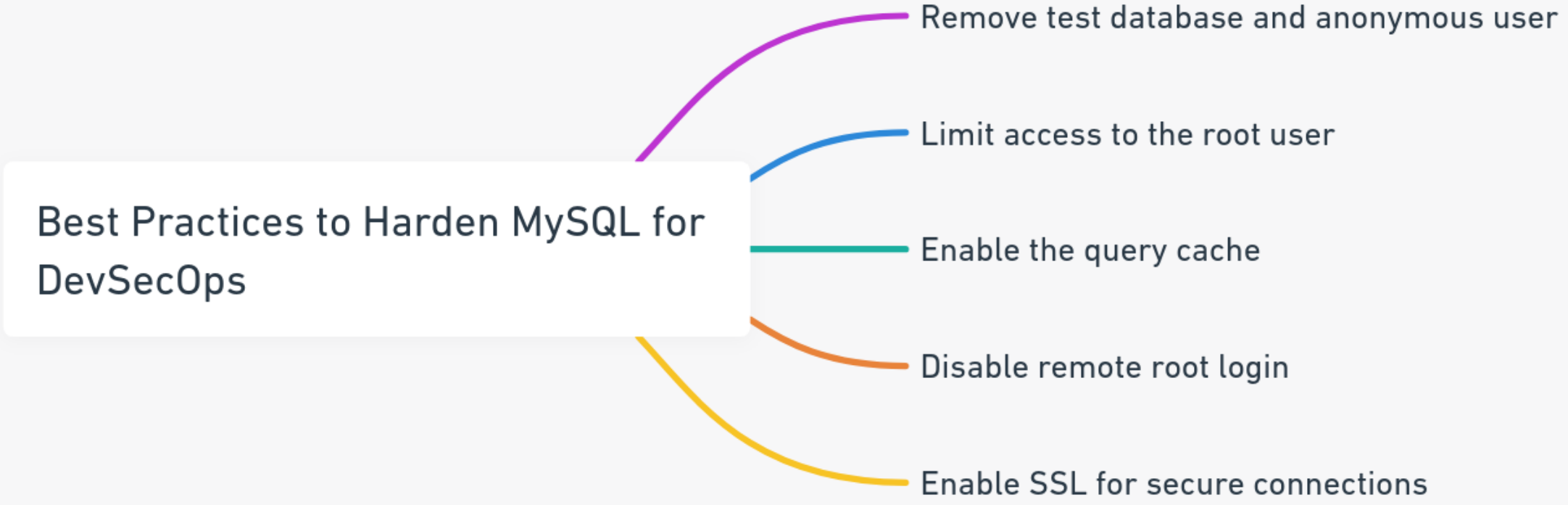
```
chown -R mongodb:mongodb /var/log/mongodb<br>chmod -R go-rwx /var/log/mongodb
```

Disable unused MongoDB features

```
sed -i '/operationProfiling:/a \ \ \ \ mode: off' /etc/mongod.conf<br>sed -i '/setParameter:/a \ \ \ \ quiet: true' /etc/mongod.conf
```

Enable firewalls and limit access to MongoDB ports

```
ufw allow from 192.168.1.0/24 to any port 27017 proto tcp<br>ufw enable
```


Best Practices to Harden MySQL for DevSecOps
Remove test database and anonymous user
Limit access to the root user
Enable the query cache
Disable remote root login
Enable SSL for secure connections

MySQL Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden MySQL for DevSecOps

Remove test database and anonymous user

```
mysql -u root -p -e "DROP DATABASE IF EXISTS test; DELETE FROM mysql.user WHERE User=''; DELETE FROM mysql.user WHERE User='root' AND Host NOT IN ('localhost', '127.0.0.1', '::1'); FLUSH PRIVILEGES;"
```

Limit access to the root user

```
mysql -u root -p -e "CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password'; GRANT ALL PRIVILEGES ON *.* TO 'newuser'@'localhost' WITH GRANT OPTION; FLUSH PRIVILEGES;"
```

Enable the query cache

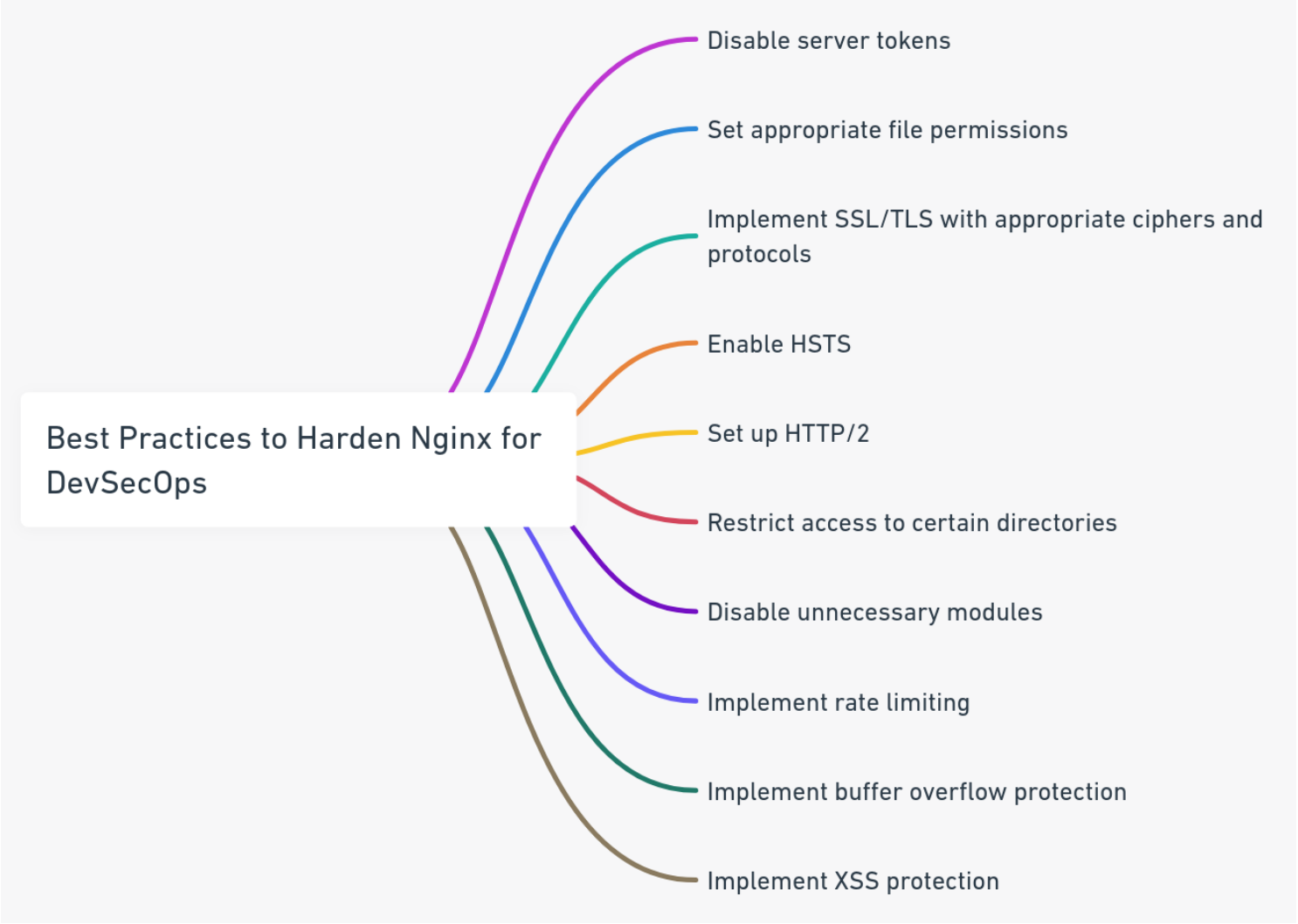
```
mysql -u root -p -e "SET GLOBAL query_cache_size = 67108864; SET GLOBAL query_cache_type = ON;"
```

Disable remote root login

Edit `/etc/mysql/mysql.conf.d/mysqld.cnf` and set `bind-address` to the IP address of the MySQL server, then restart MySQL: `systemctl restart mysql`

Enable SSL for secure connections

Edit `/etc/mysql/mysql.conf.d/mysqld.cnf` and add the following lines: `ssl-ca=/etc/mysql/certs/ca-cert.pem` `ssl-cert=/etc/mysql/certs/server-cert.pem` `ssl-key=/etc/mysql/certs/server-key.pem` Then restart MySQL: `systemctl restart mysql`


Best Practices to Harden Nginx for DevSecOps
Disable server tokens
Set appropriate file permissions
Implement SSL/TLS with appropriate ciphers and protocols
Enable HSTS
Set up HTTP/2
Restrict access to certain directories
Disable unnecessary modules
Implement rate limiting
Implement buffer overflow protection
Implement XSS protection

Nginx Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Nginx for DevSecOps

Disable server tokens

```
server_tokens off;
```

Set appropriate file permissions

`chmod 640 /etc/nginx/nginx.conf` or `chmod 440 /etc/nginx/nginx.conf` depending on your setup

Implement SSL/TLS with appropriate ciphers and protocols

`ssl_protocols TLSv1.2 TLSv1.3;`
`ssl_ciphers HIGH:!aNULL:!MD5;`

Enable HSTS

```
add_header Strict-Transport-Security "max-age=31536000; includeSubdomains; preload";
```

Set up HTTP/2

`listen 443 ssl http2;`

Restrict access to certain directories

`location /private/ { deny all; }`

Disable unnecessary modules

Comment out or remove unused modules from `nginx.conf` file.

Implement rate limiting

```
limit_req_zone $binary_remote_addr zone=mylimit:10m rate=10r/s;
```

Implement buffer overflow protection

```
proxy_buffer_size 128k;
```

```
proxy_buffers 4 256k;
```

```
proxy_busy_buffers_size 256k;
```

Implement XSS protection

```
add_header X-XSS-Protection "1; mode=block";
```


Best Practices to Harden OpenShift for DevSecOps
Disable insecure protocols and ciphers
Use `oc adm policy reconcile-cluster-role-binding`
Enable authentication and RBAC
Use `oc adm policy add-cluster-role-to-user`
Limit privileged access to the cluster
Use `oc adm policy add-scc-to-user`
Enable audit logging
Use `oc adm audit`
Enforce resource limits and quotas
Use `oc adm pod-network`
Enable network policies for isolation
Use `oc create networkpolicy`
Configure container runtime security
Use `oc adm policy add-scc-to-group`
Secure etcd and master nodes
Use `oc adm manage-node`
Regularly update and patch OpenShift components
Use `oc adm upgrade`
Enable image signing and verification
Use `oc image sign`
Use secure registry for image pull
Use `oc create secret`
Enable encryption for data in transit
Use `oc adm router`
Harden worker node security
Use `oc adm manage-node`
Implement multi-factor authentication
Use `oc adm policy`
Enable centralized logging and monitoring
Use `oc adm logs`

OpenShift Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden OpenShift for DevSecOps

Disable insecure protocols and ciphers

```
oc adm policy reconcile-cluster-role-binding
```

Enable authentication and RBAC

```
oc adm policy add-cluster-role-to-user
```

Limit privileged access to the cluster

```
oc adm policy add-scc-to-user
```

Enable audit logging

```
oc adm audit
```

Enforce resource limits and quotas

```
oc adm pod-network
```

Enable network policies for isolation

```
oc create networkpolicy
```

Configure container runtime security

```
oc adm policy add-scc-to-group
```

Secure etcd and master nodes

```
oc adm manage-node
```

Regularly update and patch OpenShift components

```
oc adm upgrade
```

Enable image signing and verification

```
oc image sign
```

Use secure registry for image pull

```
oc create secret
```

Enable encryption for data in transit

```
oc adm router
```

Harden worker node security

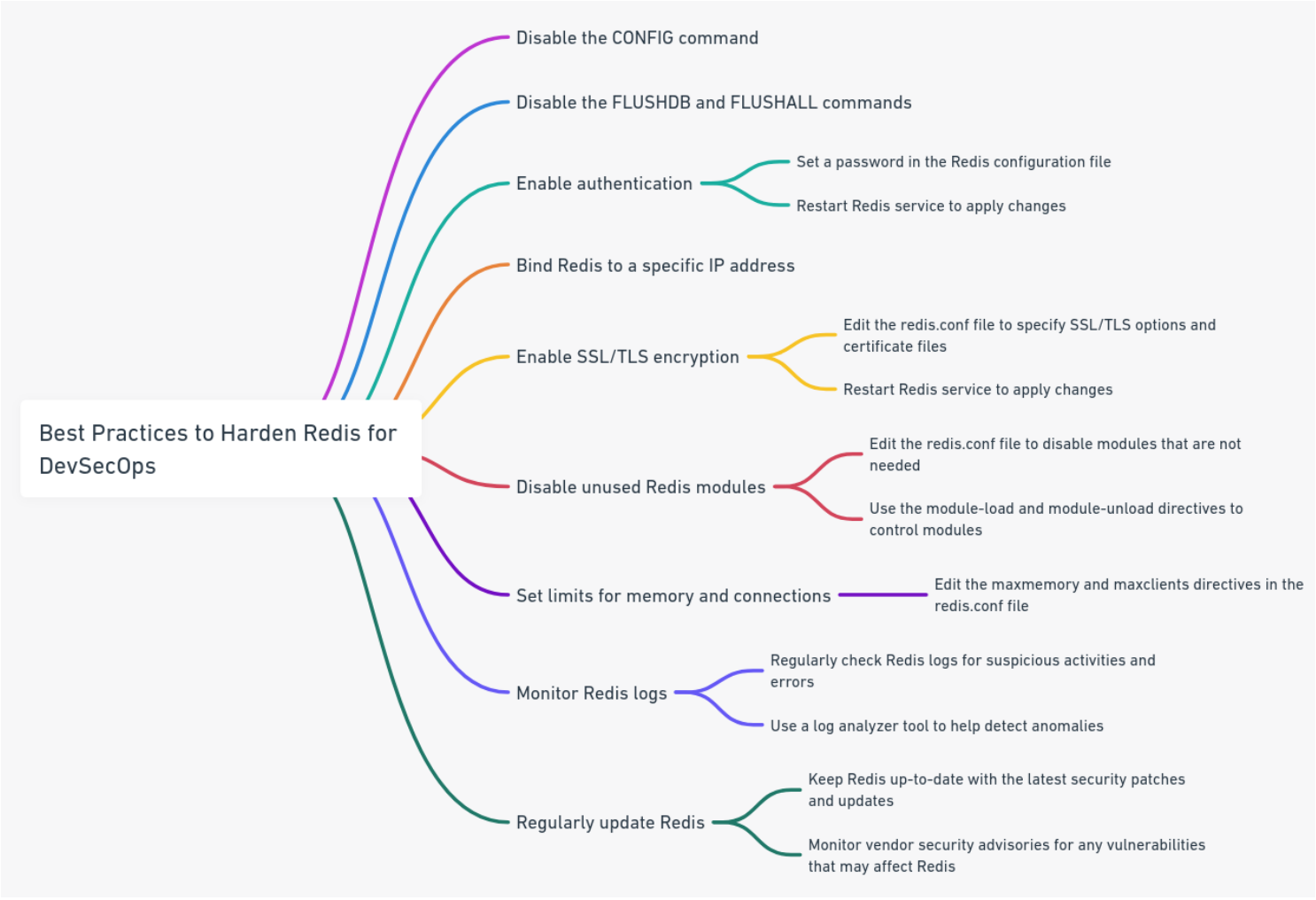
```
oc adm manage-node
```

Implement multi-factor authentication

```
oc adm policy
```

Enable centralized logging and monitoring

```
oc adm logs
```


Best Practices to Harden Redis for DevSecOps
Disable the CONFIG command
Disable the FLUSHDB and FLUSHALL commands
Enable authentication
Set a password in the Redis configuration file
Restart Redis service to apply changes
Bind Redis to a specific IP address
Enable SSL/TLS encryption
Edit the redis.conf file to specify SSL/TLS options and certificate files
Restart Redis service to apply changes
Disable unused Redis modules
Edit the redis.conf file to disable modules that are not needed
Use the module-load and module-unload directives to control modules
Set limits for memory and connections
Edit the maxmemory and maxclients directives in the redis.conf file
Monitor Redis logs
Regularly check Redis logs for suspicious activities and errors
Use a log analyzer tool to help detect anomalies
Regularly update Redis
Keep Redis up-to-date with the latest security patches and updates
Monitor vendor security advisories for any vulnerabilities that may affect Redis

Redis Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Redis for DevSecOps

Disable the CONFIG command

```
redis-cli config set config-command " "
```

Disable the FLUSHDB and FLUSHALL commands

```
redis-cli config set stop-writes-on-bgsave-error yes
```

Enable authentication

Set a password in the Redis configuration file (`redis.conf`) using the `requirepass` directive. Restart Redis service to apply changes.

Bind Redis to a specific IP address

Edit the `bind` directive in the Redis configuration file to specify a specific IP address.

Enable SSL/TLS encryption

Edit the `redis.conf` file to specify SSL/TLS options and certificate files. Restart Redis service to apply changes.

Disable unused Redis modules

Edit the `redis.conf` file to disable modules that are not needed. Use the `module-load` and `module-unload` directives to control modules.

Set limits for memory and connections

Edit the `maxmemory` and `maxclients` directives in the `redis.conf` file to set limits for Redis memory and connections.

Monitor Redis logs

Regularly check Redis logs for suspicious activities and errors. Use a log analyzer tool to help detect anomalies.

Regularly update Redis

Keep Redis up-to-date with the latest security patches and updates. Monitor vendor security advisories for any vulnerabilities that may affect Redis.


Best Practices to Harden SaltStack for DevSecOps
Generate SSL certificates for SaltStack communication
Enable SSL encryption for SaltStack communication by updating the Salt master configuration file
Disable unnecessary services and open ports
Restrict network access
Manage Salt Minion keys securely
Implement strong authentication
Secure Salt Minions
Securely distribute and manage Salt Minion keys
Disable unnecessary services and open ports on Salt Minions
Restrict network access to Salt Minions using firewalls or network ACLs
Enable authentication mechanisms, such as TLS/SSL, for secure communication
Implement strong passwords or key-based authentication for Salt Minion access
Regularly update Salt Minions to the latest stable version
Enable logging on Salt Minions and monitor logs for security events

SaltStack Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden SaltStack for DevSecOps

Generate SSL certificates for SaltStack communication

```
salt-call --local tls.create_self_signed_cert
```

Enable SSL encryption for SaltStack communication by updating the Salt master configuration file

```
# /etc/salt/master
ssl_cert: /etc/pki/tls/certs/salt.crt
ssl_key: /etc/pki/tls/private/salt.key
```

Disable unnecessary services and open ports

Disable unused services and close unnecessary ports on Salt Master and Salt Minions

Restrict network access

Configure firewalls or network ACLs to allow access only from trusted sources

Manage Salt Minion keys securely

Properly distribute, manage, and secure Salt Minion keys

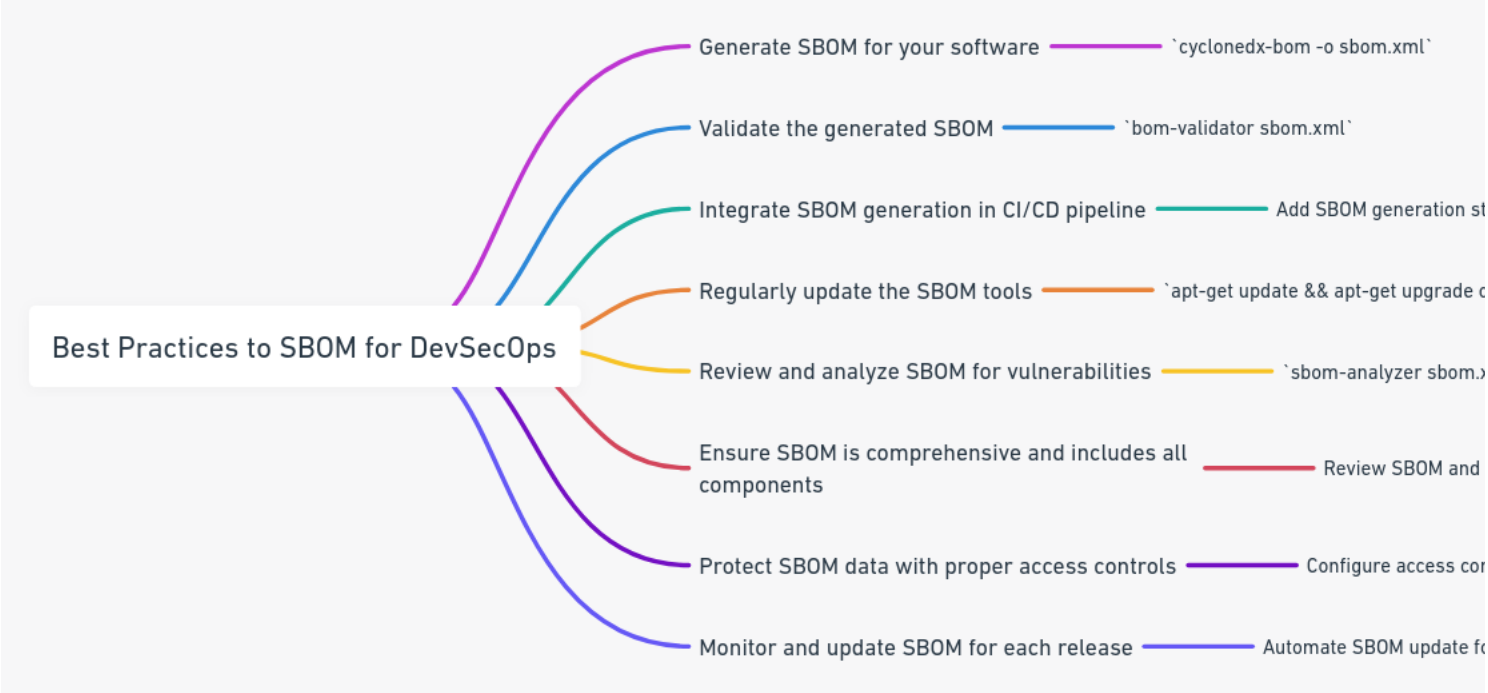
Implement strong authentication

Utilize strong passwords or key-based authentication for Salt Master and Minion access

Secure Salt Minions

- [x] Securely distribute and manage Salt Minion keys.
- [x] Disable unnecessary services and open ports on Salt Minions.
- [x] Restrict network access to Salt Minions using firewalls or network ACLs.
- [x] Enable authentication mechanisms, such as TLS/SSL, for secure communication.
- [x] Implement strong passwords or key-based authentication for Salt Minion access.
- [x] Regularly update Salt Minions to the latest stable version.

- [x] Enable logging on Salt Minions and monitor logs for security events.


Best Practices to SBOM for DevSecOps
Generate SBOM for your software
`cyclonedx-bom -o sbom.xml`
Validate the generated SBOM
`bom-validator sbom.xml`
Integrate SBOM generation in CI/CD pipeline
Add SBOM generation st
Regularly update the SBOM tools
`apt-get update && apt-get upgrade
Review and analyze SBOM for vulnerabilities
`sbom-analyzer sbom.x
Ensure SBOM is comprehensive and includes all components
Review SBOM and
Protect SBOM data with proper access controls
Configure access cor
Monitor and update SBOM for each release
Automate SBOM update fo

SBOM Security Checklist for DevSecOps

TABLE OF CONTENTS

List of some best practices to SBOM for DevSecOps

Generate SBOM for your software

```
cyclonedx-bom -o sbom.xml
```

Validate the generated SBOM

```
bom-validator sbom.xml
```

Integrate SBOM generation in CI/CD pipeline

```
Add SBOM generation step in CI/CD script
```

Regularly update the SBOM tools

```
apt-get update && apt-get upgrade cyclonedx-bom
```

Review and analyze SBOM for vulnerabilities

```
sbom-analyzer sbom.xml
```

Ensure SBOM is comprehensive and includes all components

```
Review SBOM and add missing components
```

Protect SBOM data with proper access controls

```
Configure access controls for SBOM data
```

Monitor and update SBOM for each release

```
Automate SBOM update for each release
```

Squid Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Squid for DevSecOps

Disable HTTP TRACE method

```
acl HTTP-methods method TRACE<br>http_access deny HTTP-methods
```

Limit maximum object size

```
maximum_object_size 1 MB
```

Enable access logging

```
access_log /var/log/squid/access.log
```

Limit client connections

```
acl clients src 192.168.1.0/24
```

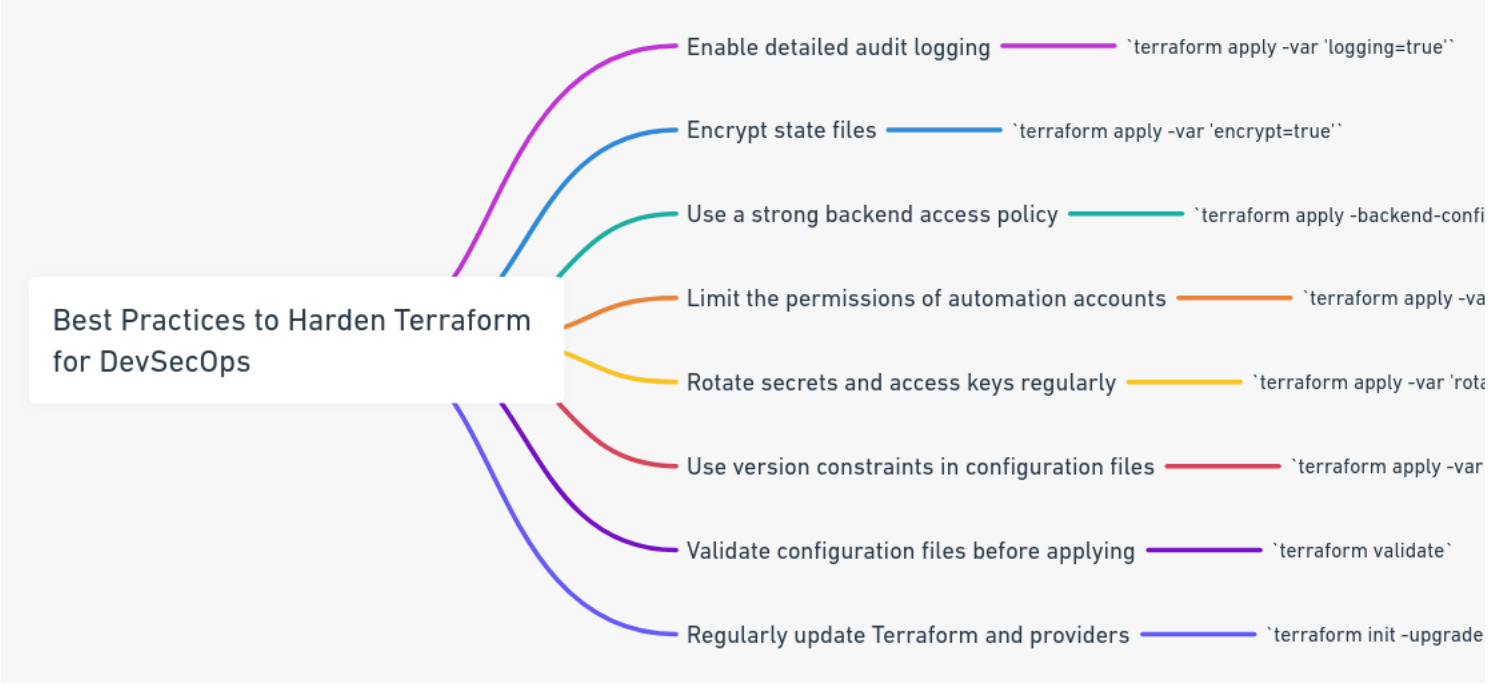
```
http_access allow clients
```

```
http_max_clients 50
```

Restrict allowed ports

```
acl Safe_ports port 80 443 8080
```

```
http_access deny !Safe_ports
```


Best Practices to Harden Terraform for DevSecOps
Enable detailed audit logging
`terraform apply -var 'logging=true'`
Encrypt state files
`terraform apply -var 'encrypt=true'`
Use a strong backend access policy
`terraform apply -backend-confi
Limit the permissions of automation accounts
`terraform apply -va
Rotate secrets and access keys regularly
`terraform apply -var 'rota
Use version constraints in configuration files
`terraform apply -var
Validate configuration files before applying
`terraform validate`
Regularly update Terraform and providers
`terraform init -upgrade

Terraform Security Checklist for DevSecOps

TABLE OF CONTENTS

List of some best practices to Terraform for DevSecOps

Enable detailed audit logging

```
terraform apply -var 'logging=true'
```

Encrypt state files

```
terraform apply -var 'encrypt=true'
```

Use a strong backend access policy

```
terraform apply -backend-config="..."
```

Limit the permissions of automation accounts

```
terraform apply -var 'permissions=limited'
```

Rotate secrets and access keys regularly

```
terraform apply -var 'rotate_secrets=true'
```

Use version constraints in configuration files

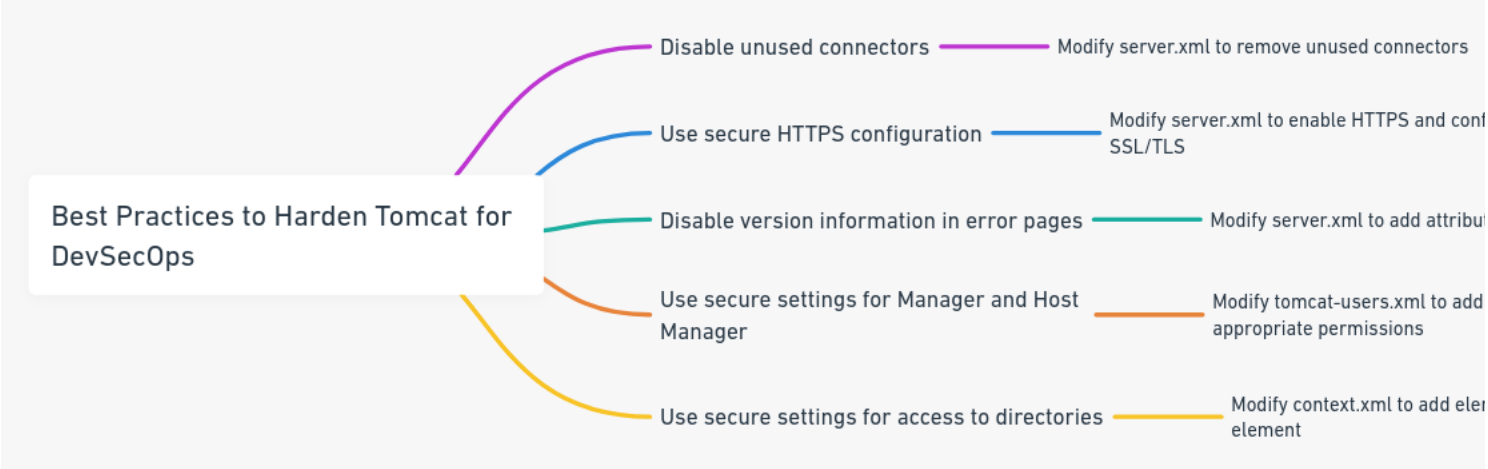
```
terraform apply -var 'version=..."
```

Validate configuration files before applying

```
terraform validate
```

Regularly update Terraform and providers

```
terraform init -upgrade
```


Best Practices to Harden Tomcat for DevSecOps
Disable unused connectors
Modify server.xml to remove unused connectors
Use secure HTTPS configuration
Modify server.xml to enable HTTPS and conf
SSL/TLS
Disable version information in error pages
Modify server.xml to add attribu
Use secure settings for Manager and Host Manager
Modify tomcat-users.xml to add
appropriate permissions
Use secure settings for access to directories
Modify context.xml to add ele
element

Tomcat Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Tomcat for DevSecOps

Disable unused connectors

Modify `server.xml` to remove the connectors not in use, e.g.:

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

Use secure HTTPS configuration

Modify `server.xml` to enable HTTPS and configure SSL/TLS, e.g.:

```
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS"
           keystoreFile="/path/to/keystore"
           keystorePass="password" />
```

Disable version information in error pages

Modify `server.xml` to add the following attribute to the `<Host>` element:

```
errorReportValveClass="org.apache.catalina.valves.ErrorReportValve" showReport="false" showServerInfo="false"
```

Use secure settings for Manager and Host Manager

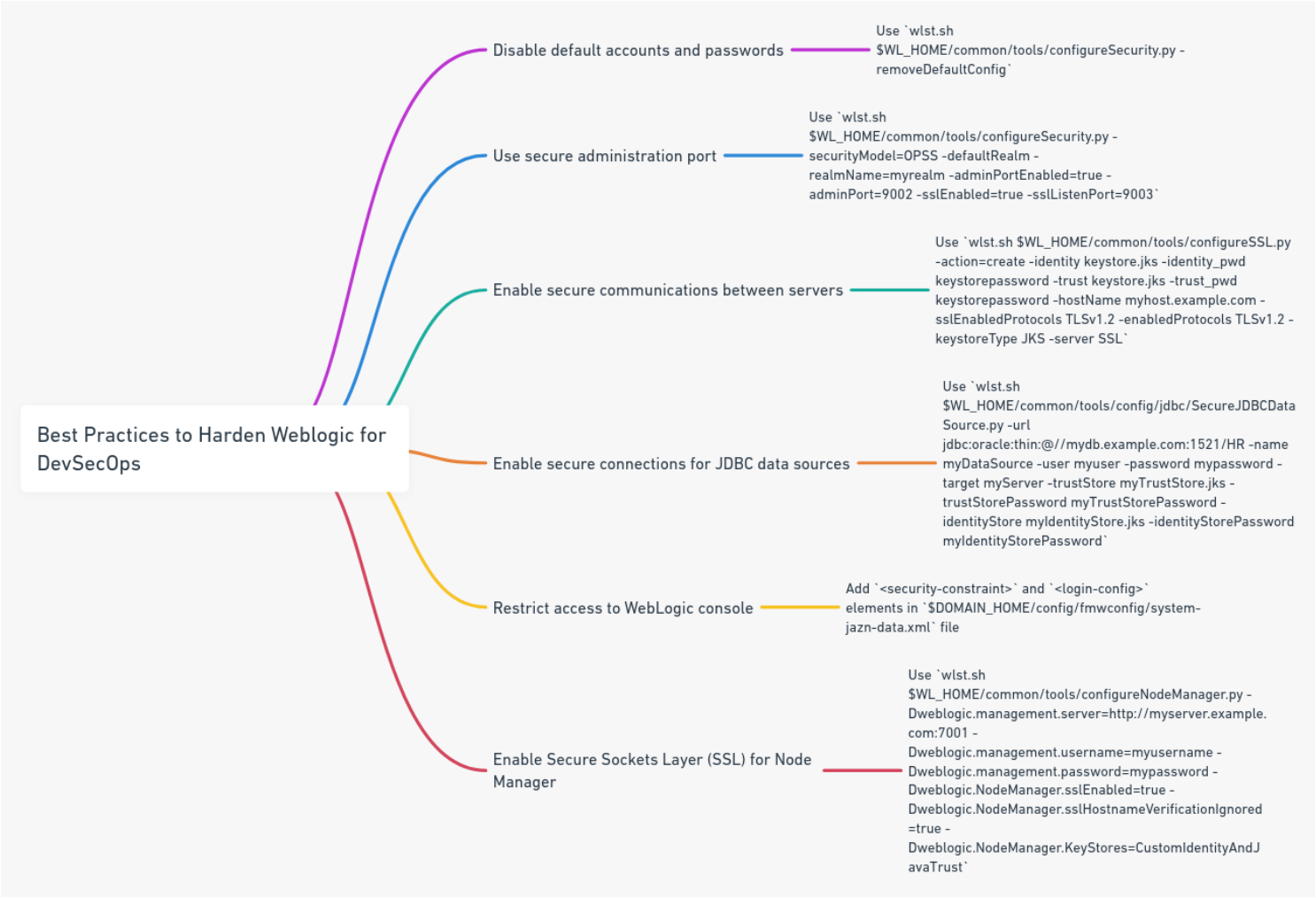
Modify `tomcat-users.xml` to add roles and users with the appropriate permissions, e.g.:

```
<role rolename="manager-gui"/>
<user username="tomcat" password="password" roles="manager-gui"/>
```

Use secure settings for access to directories

Modify `context.xml` to add the following element to the `<Context>` element:

```
  <Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127\.0\.0\.1|192\.168\.0\.\d+"/>
```


Best Practices to Harden Weblogic for DevSecOps
Disable default accounts and passwords
Use `wlst.sh $WL_HOME/common/tools/configureSecurity.py -removeDefaultConfig`
Use secure administration port
Use `wlst.sh $WL_HOME/common/tools/configureSecurity.py -securityModel=OPSS -defaultRealm -realmName=myrealm -adminPortEnabled=true -adminPort=9002 -sslEnabled=true -sslListenPort=9003`
Enable secure communications between servers
Use `wlst.sh $WL_HOME/common/tools/configureSSL.py -action=create -identity keystore.jks -identity_pwd keystorepassword -trust keystore.jks -trust_pwd keystorepassword -hostName myhost.example.com -sslEnabledProtocols TLSv1.2 -enabledProtocols TLSv1.2 -keystoreType JKS -server SSL`
Enable secure connections for JDBC data sources
Use `wlst.sh $WL_HOME/common/tools/config/jdbc/SecureJDBCDataSource.py -url jdbc:oracle:thin:@//mydb.example.com:1521/HR -name myDataSource -user myuser -password mypassword -target myServer -trustStore myTrustStore.jks -trustStorePassword myTrustStorePassword -identityStore myIdentityStore.jks -identityStorePassword myIdentityStorePassword`
Restrict access to WebLogic console
Add `<security-constraint>` and `<login-config>` elements in `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml` file
Enable Secure Sockets Layer (SSL) for Node Manager
Use `wlst.sh $WL_HOME/common/tools/configureNodeManager.py -Dweblogic.management.server=http://myserver.example.com:7001 -Dweblogic.management.username=myusername -Dweblogic.management.password=mypassword -Dweblogic.NodeManager.sslEnabled=true -Dweblogic.NodeManager.sslHostnameVerificationIgnored=true -Dweblogic.NodeManager.KeyStores=CustomIdentityAndJavaTrust`

Weblogic Hardening for DevSecOps

TABLE OF CONTENTS

List of some best practices to harden Weblogic for DevSecOps

Disable default accounts and passwords

```
wlst.sh $WL_HOME/common/tools/configureSecurity.py -removeDefaultConfig
```

Use secure administration port

```
wlst.sh $WL_HOME/common/tools/configureSecurity.py -securityModel=OPSS -defaultRealm -realmName=myrealm -adminPortEnabled=true -adminPort=9002 -
sslEnabled=true -sslListenPort=9003
```

Enable secure communications between servers

```
wlst.sh $WL_HOME/common/tools/configureSSL.py -action=create -identity keystore.jks -identity_pwd keystorepassword -trust keystore.jks -trust_pwd
keystorepassword -hostName myhost.example.com -sslEnabledProtocols TLSv1.2 -enabledProtocols TLSv1.2 -keystoreType JKS -server SSL
```

Enable secure connections for JDBC data sources

```
wlst.sh $WL_HOME/common/tools/config/jdbc/SecureJDBCDataSource.py -url jdbc:oracle:thin:@//mydb.example.com:1521/HR -name myDataSource -user
myuser -password mypassword -target myServer -trustStore myTrustStore.jks -trustStorePassword myTrustStorePassword -identityStore
myIdentityStore.jks -identityStorePassword myIdentityStorePassword
```

Restrict access to WebLogic console

Add `<security-constraint>` and `<login-config>` elements in `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml` file

Enable Secure Sockets Layer (SSL) for Node Manager

```
wlst.sh $WL_HOME/common/tools/configureNodeManager.py -Dweblogic.management.server=http://myserver.example.com:7001 -
Dweblogic.management.username=myusername -Dweblogic.management.password=mypassword -Dweblogic.NodeManager.sslEnabled=true -
Dweblogic.NodeManager.sslHostnameVerificationIgnored=true -Dweblogic.NodeManager.KeyStores=CustomIdentityAndJavaTrust
```