Kubernetes: Security Basics

Contacts

Péter Megyesi

megyesi@leannet.eu

twitter.com/M3gy0

linkedin.com/in/M3gy0

Dávid Szabó

szabo@leannet.eu

twitter.com/szabvid

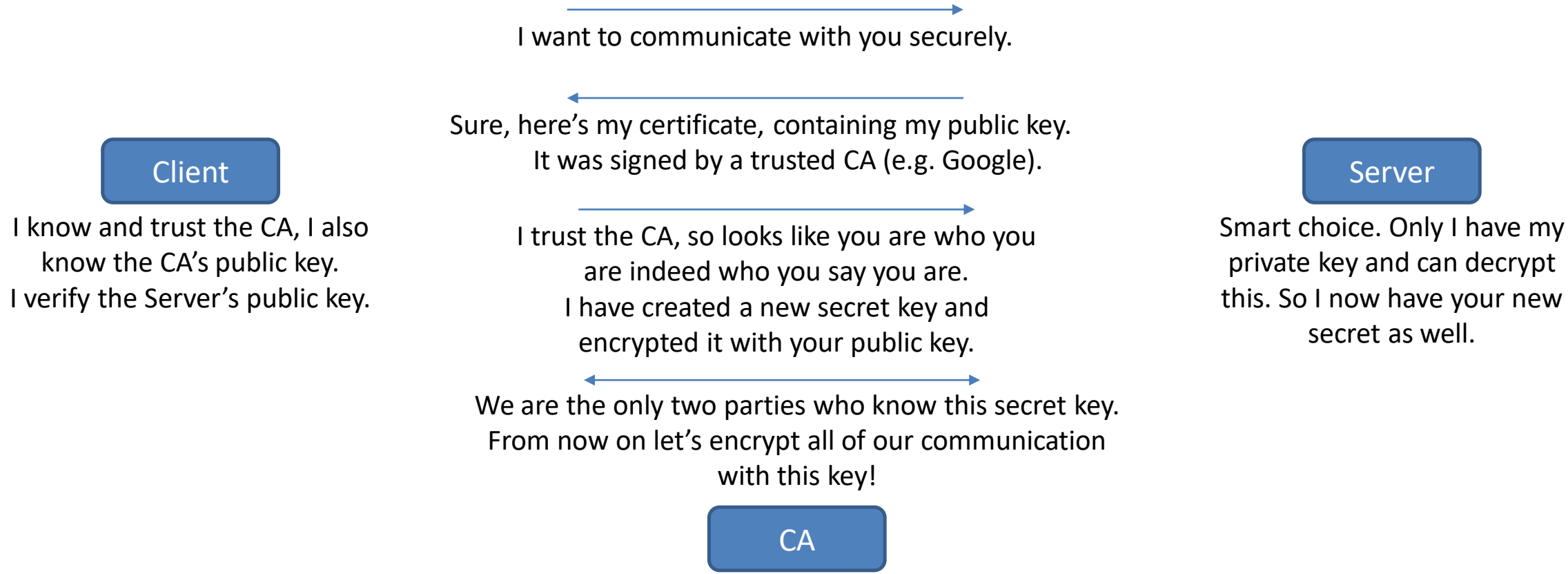
linkedin.com/in/szabvid

Course Outline

1. What is Kubernetes?
   - Components
   - Installation
2. Basics of Docker
   - Namespaces
   - Building and running Docker images
3. Pods and Deployments
   - Running basic workloads in Kubernetes
   - Scale, Update, Rollback
4. Advanced Pod configuration
   - Args, Envs, ConfigMaps, Secrets
   - Init- and sidecar containers
   - Scheduling and debugging
5. Networking in Kubernetes
   - What are network plugins?
   - Service abstraction and ingress
6. Persistent storage
   - Basics of storage: block vs. object vs. file system
   - StoragesClass, PVC, PV
7. **Security**
   - **RBAC: Roles, ServiceAccounts, RoleBindings**
   - **Security context and network security policy**
8. Advanced topics
   - Helm
   - Custom resources and operators

Some very basic terms

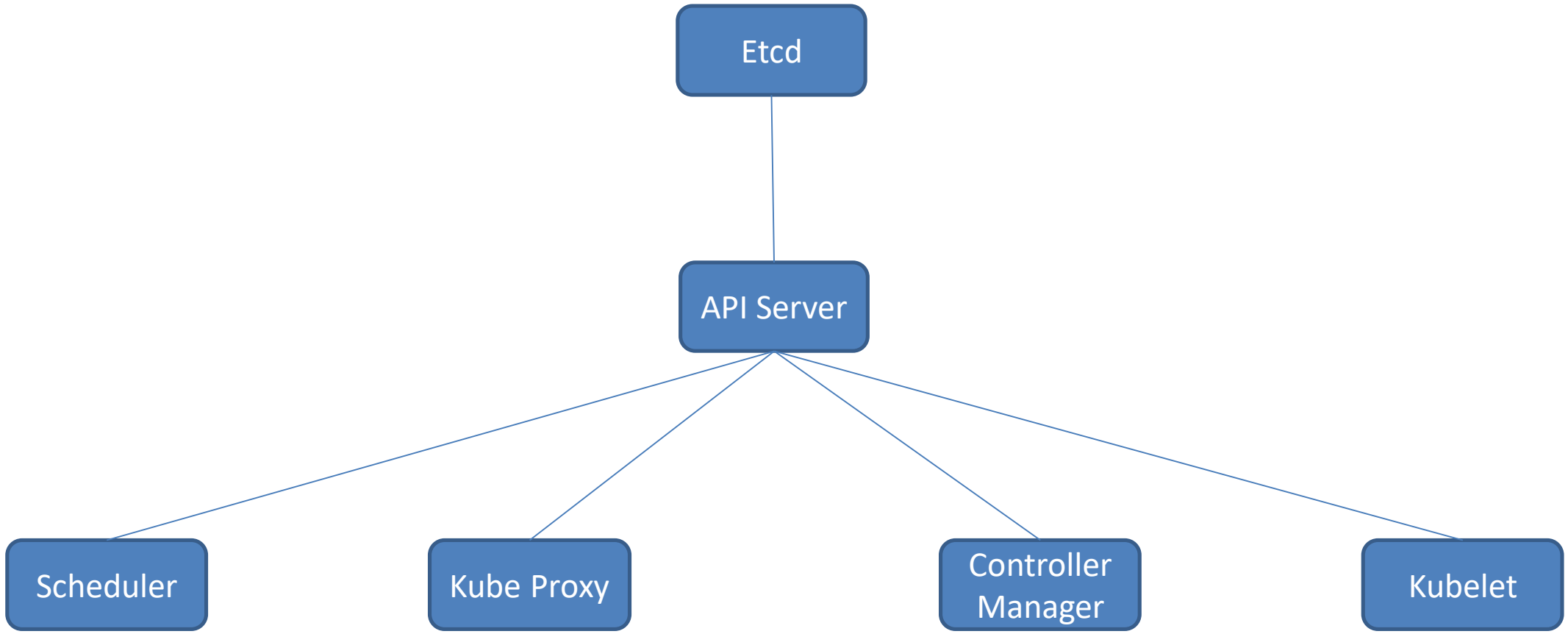
- Public key cryptography & signature axioms:
  1. Any message encrypted by Bob's public key can only be decrypted by Bob's private key.
  2. Anyone with access to Alice's public key can verify that a message (signature) could only have been created by someone with access to Alice's private key.



Some very basic terms

- Certificate Authority (CA): a 3rd party that everyone trusts
- Certificate signing request (CSR): message sent from an applicant to a CA to get a certificate
  - Contains public key, identity information, integrity protection (digital signature).

Kubernetes Public Key Infrastructure (PKI)



Kubernetes Public Key Infrastructure (PKI)

- Certificate Authority (CA):
  - Cluster CA
  - All cluster certs are signed by it
  - Used by components (Scheduler, ApiServer, etc.) to validate the other (Kubelet, etc.)
- Two types of certificates:
  - Serving certificate (server needs to have it):
    - For others to authenticate the server
      - E.g. API Server and Kubelet has their API as well
        - » Kubelet's API is consumed by the API Server when getting logs, metrics, exec, etc.
        - » Serving certificate and key are required for HTTPS
        - » API Server can authenticate/validate the Kubelet
  - Client certificate (client needs to have it):
    - For others to authenticate the client
      - E.g. API Server also needs a client certificate in order the Kubelet be able to authenticate it
        - » since it does not want to serve anybody

Kubernetes Public Key Infrastructure (PKI)

Demo

- Kubernetes uses X.509 Client Cert Authentication:
  - X.509 is a standard defining the format of public key certificates
  - Strategy to authenticate request
  - Any request that presents a client certificate signed by the Cluster CA is authenticated
  - User is obtained from Common Name (CN) field
  - Groups are obtained from Organization (O) field:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 3267605220748972702 (0x2d58dda2c0f4529e)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=kubernetes
        Validity
            Not Before: Aug 19 16:49:36 2020 GMT
            Not After : Aug 19 18:46:01 2021 GMT
        Subject: O=system:masters, CN=kube-apiserver-kubelet-client
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ca:bf:04:b2:8b...
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Client Authentication
    Signature Algorithm: sha256WithRSAEncryption
         c8:03:2b:f1:33...
-----BEGIN CERTIFICATE-----
MIIC/zCCAe...jzua
-----END CERTIFICATE-----
```

Kubernetes Public Key Infrastructure (PKI)

- Kubernetes uses X.509 Client Cert Authentication:
  - Strategy to authenticate request
  - Any request that presents a client certificate signed by the Cluster CA is authenticated
  - User is obtained from Common Name (CN) field
  - Groups are obtained from Organization (O) field:
- Each core component has a client certificate

Component	Common Name	Organizations
Controller Manager	system:kube-controller-manager	
Scheduler	system:kube-scheduler	
Kube Proxy	system:kube-proxy	
Kubelet	system:node:$(hostname)	system:nodes

```
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 1836420361979251427 (0x197c4696e6114ae3)
  Signature Algorithm: sha256WithRSAEncryption
    Issuer: CN=kubernetes
    Validity
      Not Before: Aug 19 16:49:36 2020 GMT
      Not After : Aug 19 18:46:01 2021 GMT
    Subject: CN=kube-apiserver
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
          00:cb:c1:f0:62...
        Exponent: 65537 (0x10001)
    X509v3 extensions:
      X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
      X509v3 Extended Key Usage:
        TLS Web Server Authentication
      X509v3 Subject Alternative Name:
        DNS:k8s-master-1-13, DNS:kubernetes, DNS:kubernetes.default,
        DNS:kubernetes.default.svc, DNS:kubernetes.default.svc.cluster.local, IP
        Address:10.96.0.1, IP Address:10.156.0.20
  Signature Algorithm: sha256WithRSAEncryption
    4c:14:63:17...
-----BEGIN CERTIFICATE-----
MIIDYDCCAki...EPfUsAtvGvX+Fpy5/XCJA==
-----END CERTIFICATE-----
```

Kubernetes Public Key Infrastructure (PKI)

- Which exact certificates the components has?
  - Check config file usually in `/etc/kubernetes`
  - Check with `ps aux | grep <COMPONENT NAME>`

- command:
  - kube-apiserver
  - --authorization-mode=Node,RBAC
  - --advertise-address=10.156.0.20
  - --allow-privileged=true
  - **--client-ca-file=/etc/kubernetes/pki/ca.crt**
  - --enable-admission-plugins=NodeRestriction
  - --enable-bootstrap-token-auth=true
  - **--etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt**
  - **--etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt**
  - **--etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key**
  - --etcd-servers=https://127.0.0.1:2379
  - --insecure-port=0
  - **--kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt**
  - **--kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key**
  - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
  - **--proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt**
  - **--proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key**
  - --requestheader-allowed-names=front-proxy-client
  - **--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt**
  - --requestheader-extra-headers-prefix=X-Remote-Extra-
  - --requestheader-group-headers=X-Remote-Group
  - --requestheader-username-headers=X-Remote-User
  - --secure-port=6443
  - --service-account-key-file=/etc/kubernetes/pki/sa.pub
  - --service-cluster-ip-range=10.96.0.0/12
  - **--tls-cert-file=/etc/kubernetes/pki/apiserver.crt**
  - **--tls-private-key-file=/etc/kubernetes/pki/apiserver.key**

Kubernetes Public Key Infrastructure (PKI)

- Which exact certificates the components has?
  - Check config file usually in `/etc/kubernetes`
  - Check with `ps aux | grep <COMPONENT NAME>`

- command:
  - kube-controller-manager
  - --address=127.0.0.1
  - --allocate-node-cidrs=true
  - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
  - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
  - **--client-ca-file=/etc/kubernetes/pki/ca.crt**
  - --cluster-cidr=10.244.0.0/16
  - **--cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt**
  - **--cluster-signing-key-file=/etc/kubernetes/pki/ca.key**
  - --controllers=*,bootstrapsigner,tokencleaner
  - --kubeconfig=/etc/kubernetes/controller-manager.conf
  - --leader-elect=true
  - --node-cidr-mask-size=24
  - **--requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt**
  - **--root-ca-file=/etc/kubernetes/pki/ca.crt**
  - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
  - --use-service-account-credentials=true

Kubernetes Public Key Infrastructure (PKI)

- Which exact certificates the components has?
  - Check config file usually in `/etc/kubernetes`
  - Check with `ps aux | grep <COMPONENT NAME>`

- command:
  - etcd
  - --advertise-client-urls=https://10.156.0.20:2379
  - **--cert-file=/etc/kubernetes/pki/etcd/server.crt**
  - --client-cert-auth=true
  - --data-dir=/var/lib/etcd
  - --initial-advertise-peer-urls=https://10.156.0.20:2380
  - --initial-cluster=k8s-master-1-13=https://10.156.0.20:2380
  - **--key-file=/etc/kubernetes/pki/etcd/server.key**
  - --listen-client-urls=https://127.0.0.1:2379,https://10.156.0.20:2379
  - --listen-peer-urls=https://10.156.0.20:2380
  - --name=k8s-master-1-13
  - **--peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt**
  - --peer-client-cert-auth=true
  - **--peer-key-file=/etc/kubernetes/pki/etcd/peer.key**
  - **--peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt**
  - --snapshot-count=10000
  - **--trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt**

Kubernetes Public Key Infrastructure (PKI)

- Kubelet is a bit different (it is not a control plane component):
  - Have to be identified for each Node separately
    - Otherwise every kubelet could initiate writes/reads for other Nodes as well
  - It certificates are stored in different place (`/var/lib/kubelet`)
  - For Kubelet there is a Cert Bootstrapping Step:
    - Kubelet certs can be automatically signed:
      1. Kubelet creates CSR using Bootstrap token
      2. CSRApprovingController approves the CSR automatically
      3. CSRSignerController sings the CSR
      4. Kubelet downloads the generated certificate and starts using it
  - Kubelet can request new client certificate before it expires (Beta)
  - Kubelet can also rotate the serving certificate (Alpha, must be enabled by feature flag)

Kubelet is a bit different:

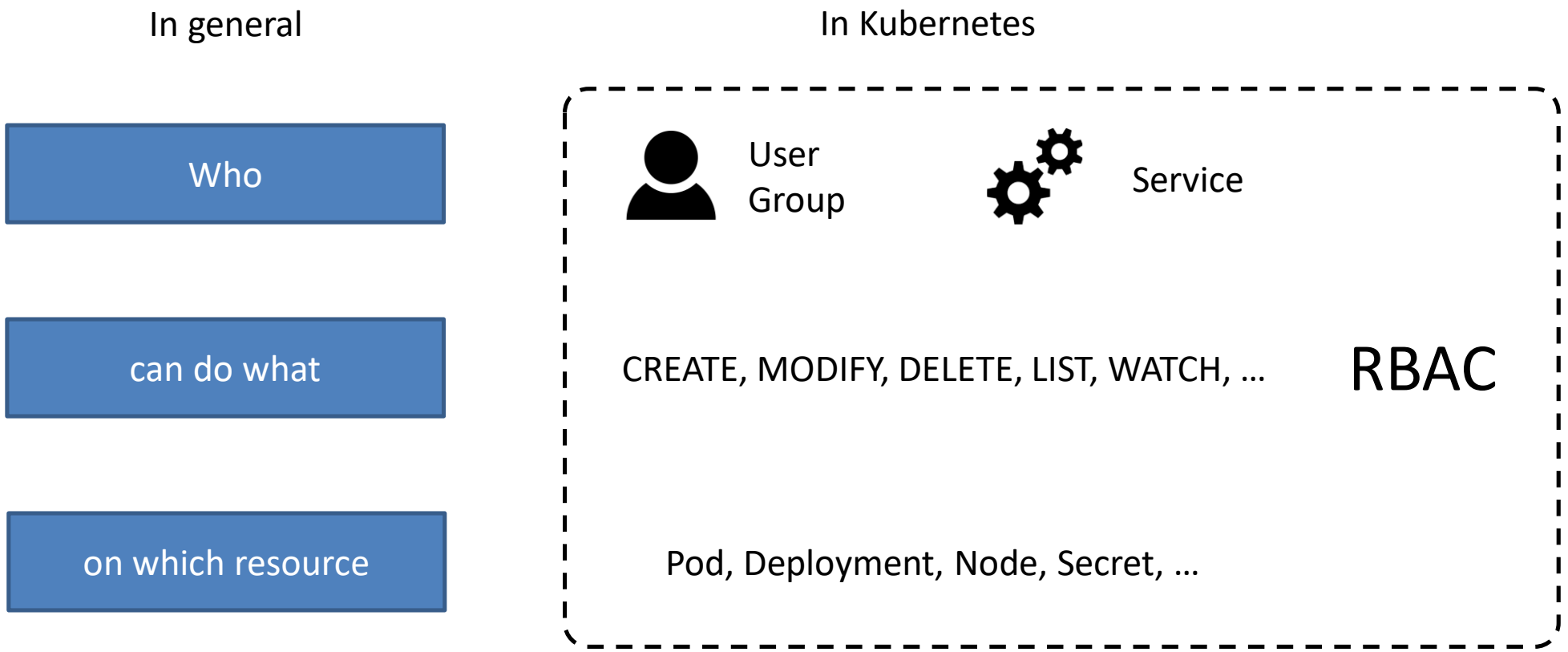
```
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
```

All the Certificates that are used in a Kubernetes Cluster

Component	Certificate	Purpose
API Server	Cluster CA	Authenticate clients
API Server	Etcd CA	Etcd server authentication
API Server	Etcd client cert	Etcd client authentication
API Server	Serving cert	Serving API over HTTPS
API Server	Kubelet client cert	Authenticate against Kubelet
Controller Manager	Client cert	Authenticate against API Server
Controller Manager	Cluster CA	Embedding in service account secrets
Scheduler	Client cert	Authenticate against API Server
Kubelet	Serving cert	Serving API over HTTPS
Kubelet	Client cert	Authenticate against API Server
Kubelet	Cluster CA	Authenticate clients
Kube Proxy	Client cert	Authenticate against API Server

IAM in Kubernetes

- IAM = Identity and Access Management

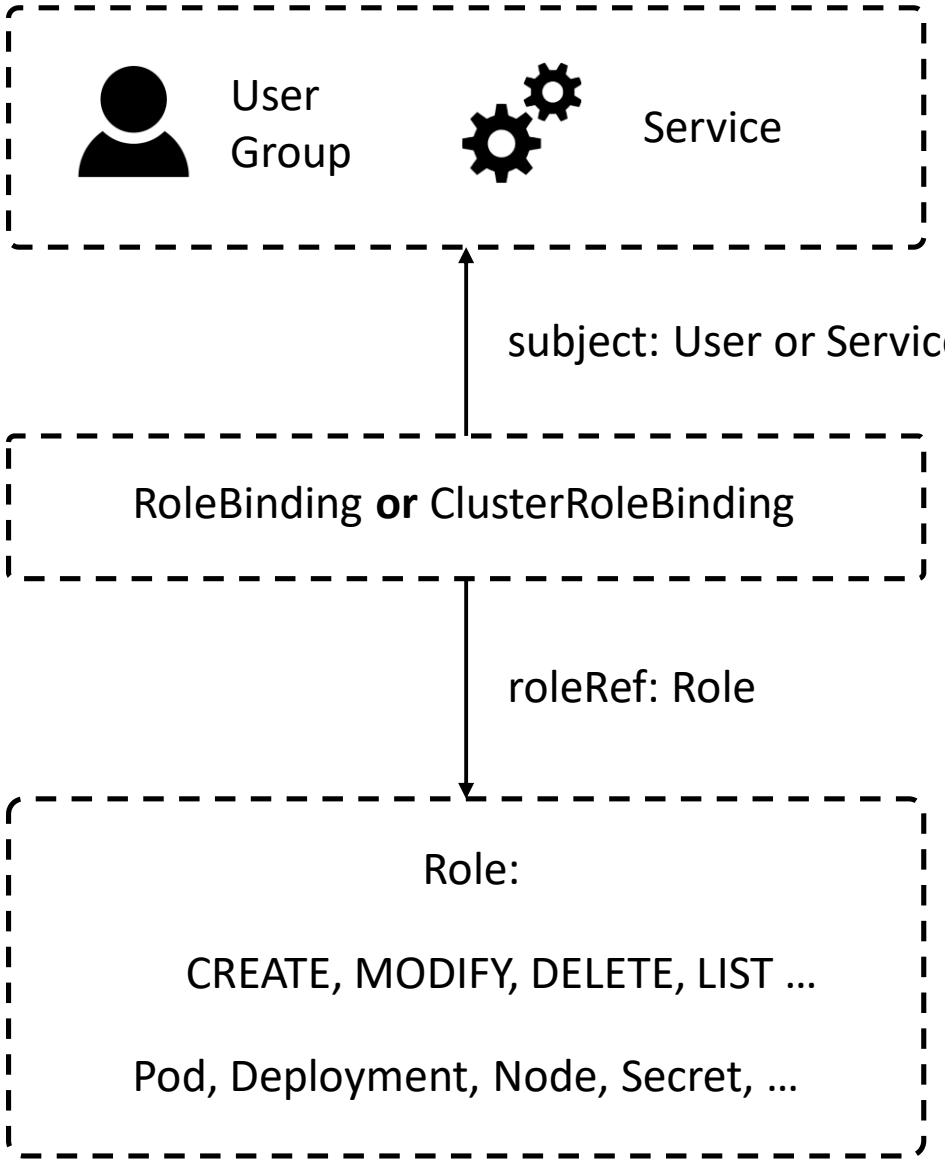


IAM in Kubernetes - WHO

- User:
  - It is assumed that a cluster-independent service manages normal users in the following ways:
    - an administrator distributing private keys
    - a user store like Keystone or Google Accounts
    - a file with a list of usernames and passwords
  - In this regard, *Kubernetes does not have objects which represent normal user accounts.* Normal users cannot be added to a cluster through an API call.
  - Any user that presents a valid certificate signed by the cluster's certificate authority (CA) is considered authenticated. Username is the common name field in the 'subject' of the cert (e.g., "/CN=bob").
  - After authentication RBAC determines “can do what on which resources” --> Authorization
- ServiceAccount:
  - Kubernetes objects crated by the API Server
  - Has a token (stored in a Secret) that can be used to authenticate with Bearer token against the API Server
  - SA is associated with pods running in the cluster
    - Bearer tokens are mounted into pods and allow in-cluster processes to talk to the API server
  - After authentication RBAC determines “can do what on which resources” --> Authorization

IAM in Kubernetes – a deeper look at RBAC

WHO



- RoleBinding: to namespace
- ClusterRoleBinding: clusterwide
- Interchangable but not the same!

WHAT ON WHICH RESOURCES

IAM in Kubernetes – a deeper look at RBAC

Demo

WHO

WHAT ON WHICH RESOURCES

User
Group

Service

subjects

RoleBinding **or**
ClusterRoleBinding

roleRef

Role:

CREATE, MODIFY, DELETE, LIST …

Pod, Deployment, Node, Secret, …

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: monitoring-sa
  namespace: monitoring
```

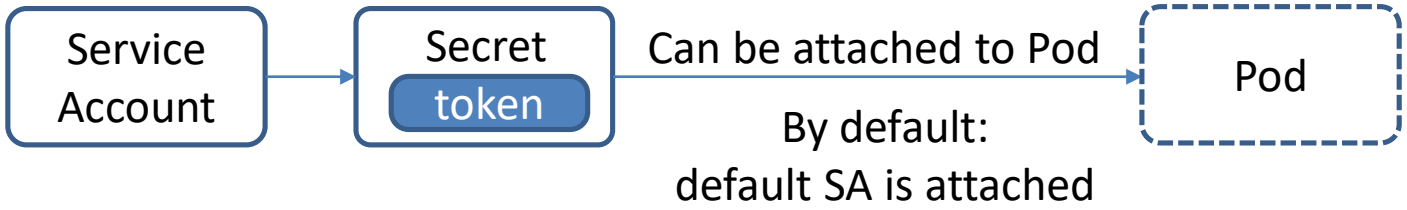
```
kind: RoleBinding
apiVersion: ...
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: ServiceAccount
  name: monitoring-sa
  namespace: monitoring
roleRef:
  kind: Role or ClusterRole
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

```
kind: Role
apiVersion: ...
metadata:
  namespace: monitoring
  name: pod-reader
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - watch
  - list
```

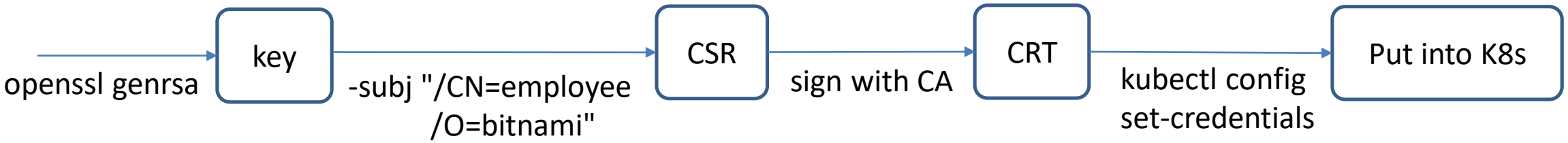
IAM in Kubernetes – a deeper look at RBAC

Demo

- Creating a service account:
  - `kubectl create serviceaccount my-logger --namespace logging`



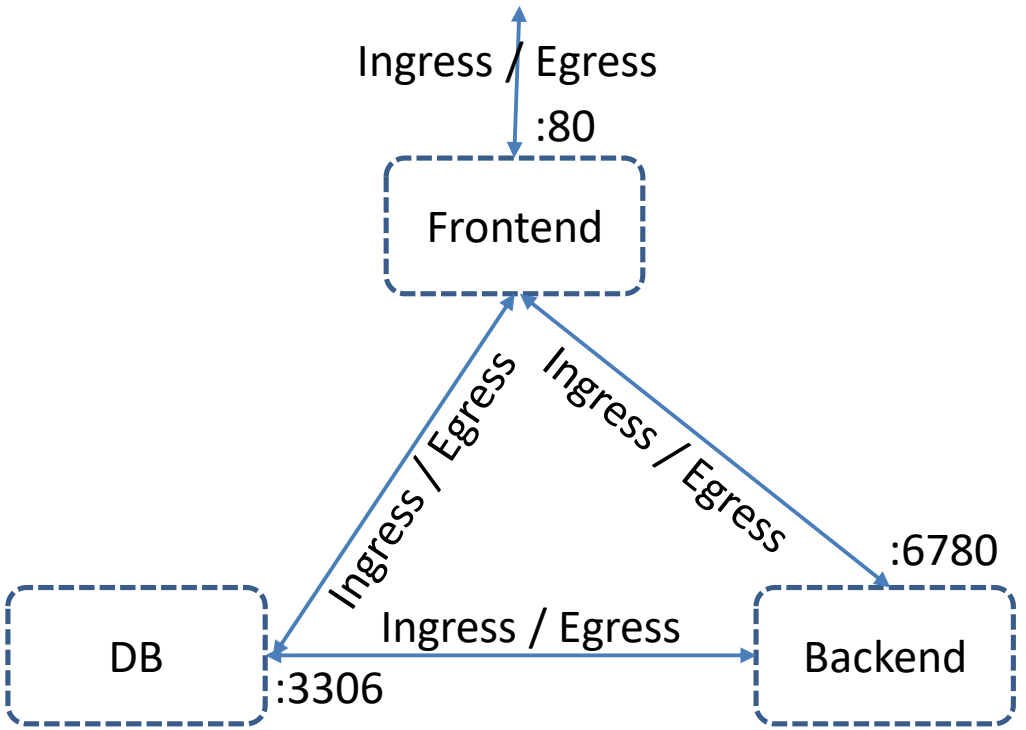
- Creating user (with openssl):



At this point we can assign roles with (cluster) role bindings to have rights to do something with resources

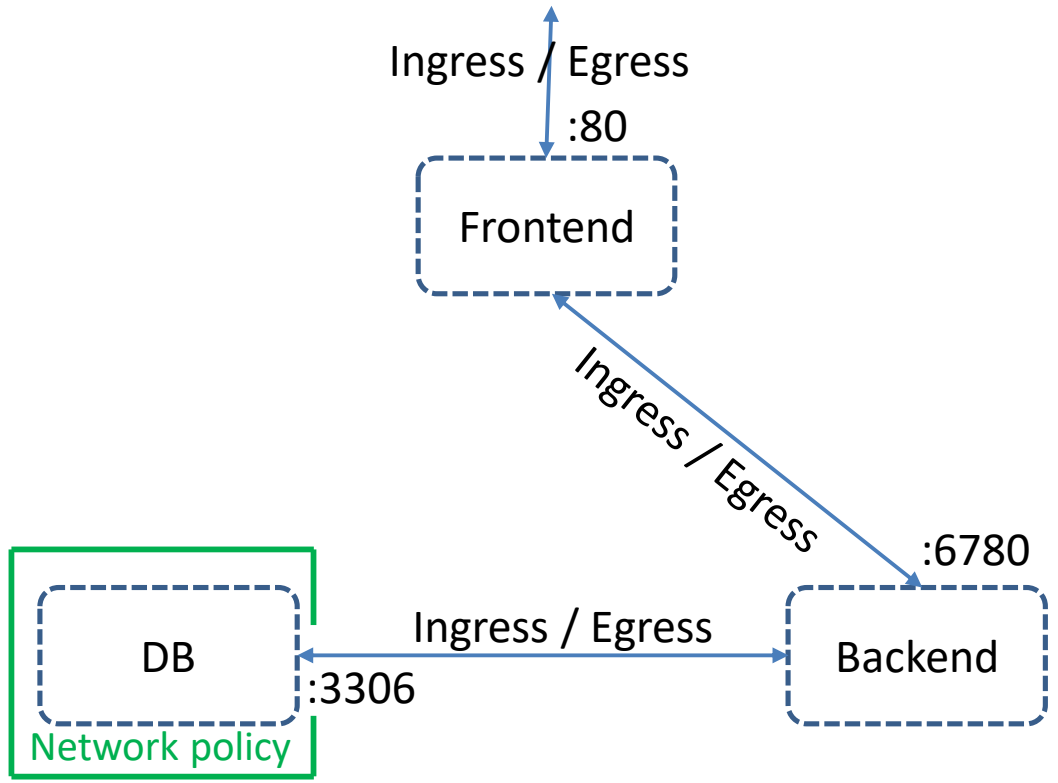
Network Policies

- Every Pod can have default ingress and egress traffic
- In K8s by default every pod can reach each other:
  - Which is not too secure…
- Network policies can help to define rules:
  - Very sophisticated:
    - Ingress / Egress
    - IpBlock
    - namespaceSelector
    - podSelector
    - ports



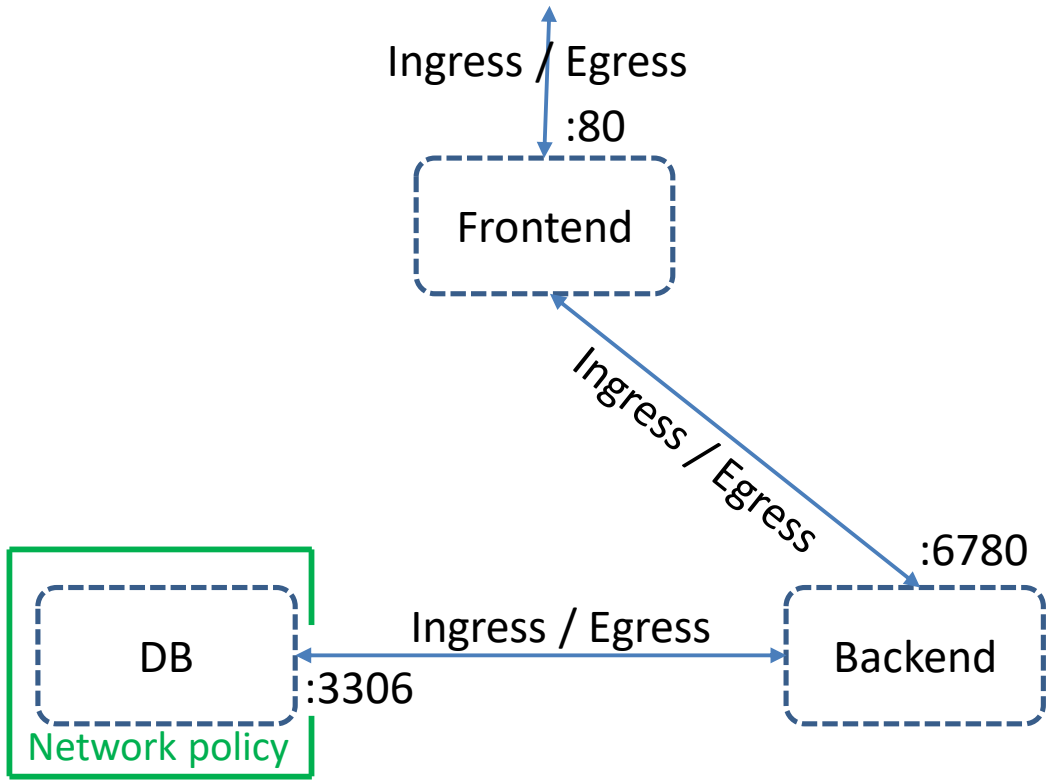
Network Policies

- Every Pod can have default ingress and egress traffic
- In K8s by default every pod can reach each other:
  - Which is not too secure…
- Network policies can help to define rules:
  - Very sophisticated:
    - Ingress / Egress
    - IpBlock
    - namespaceSelector
    - podSelector
    - ports
  - E.g.
    - DB: Allow ingress traffic from only backend on port 3306



Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: db-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: backend
    ports:
    - protocol: TCP
      port: 3306
```



Not all CNI plugin support Network Policies, e.g. Flannel:

- **Still can be defined, however, won't have any effect!**

Sealed Secrets

- It’s nice to store every configuration (since it is part of the infrastructure) in Git:
  - Related topics are: GitOps, Infrastructue-as-a-Code
- However, storing sensitive information as a base64 string is not secure.
  - Thus Secret manifest file cannot be stored in a source-code repository
- SealedSecret can help with this:
  - Developed by Bitnami Labs and is open-source
  - The idea:
    - Have a cluster-side Kubernetes controller/operator and a client-side utility called `kubeseal` (simple binary)
    - seal K8s Secrets with `kubeseal` using the asymmetric crypto algorithm: Secret --> SealedSecret
    - SealedSecrets are Kubernetes resources that contain encrypted Secrets
    - SealedSecrets can ONLY be decrypted by the controller deployed in the cluster
- Workflow: create Secrets locally --> seal them with `kubeseal` --> store only SealedSecrets in Git

Some other considerations (best practices)

- Running Containers as a Non-Root User:

```
containers:
- name: demo
  image: cloudnatived/demo:hello
  securityContext:
    runAsUser: 1000
```

- Setting a Read-Only Filesystem:

```
containers:
- name: demo
  image: cloudnatived/demo:hello
  securityContext:
    readOnlyRootFilesystem: true
```

- Use Capabilities:

```
containers:
- name: demo
  image: cloudnatived/demo:hello
  securityContext:
    capabilities:
      drop: ["CHOWN", "NET_RAW", "SETPCAP"]
      add: ["NET_ADMIN"]
```

Some other considerations (best practices)

- Make sure RBAC is enabled in all your clusters.
  - If your app needs access to cluster resources, create a service account for it with minimal permissions.
- Guard Access to Cluster-Admin role:
  - Don’t fix problems by granting *cluster-admin* unnecessarily (even when you read it on StackOverflow)
- Scanning container images:
  - Don’t run containers from untrusted sources or when you’re not sure what’s in them.
  - Run a scanning tool like Clair or MicroScanner:
    - Even on yours as well
- Use monitoring tools (K8s Dasboard, Stackdriver, …) but with minimum privileges
  - Never expose them to the Internet
  - Instead access it via kubectl proxy