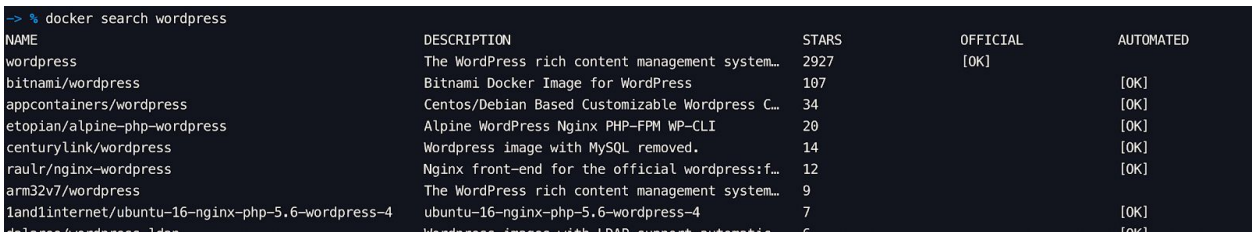
Summary

Kubernetes Security

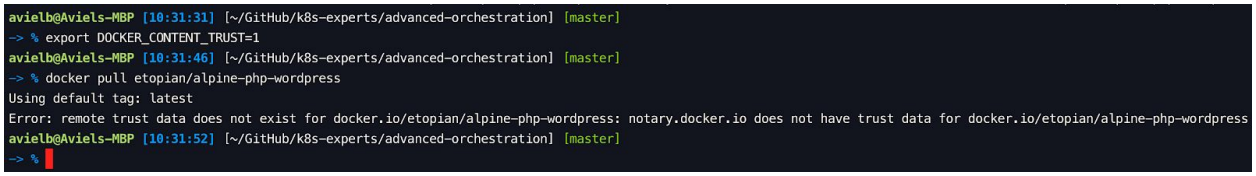
Securing Docker Images

- Choose third-party containers carefully
  With Docker, you can pull down containers from public repositories. This means you are placing your trust in whoever created the container.



```
-> % docker search wordpress
NAME                                              DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
wordpress                                         The WordPress rich content management system…   2927                [OK]
bitnami/wordpress                                 Bitnami Docker Image for WordPress              107                                     [OK]
appcontainers/wordpress                           Centos/Debian Based Customizable Wordpress C…   34                                      [OK]
etopian/alpine-php-wordpress                      Alpine WordPress Nginx PHP-FPM WP-CLI           20                                      [OK]
centurylink/wordpress                             Wordpress image with MySQL removed.             14                                      [OK]
raulr/nginx-wordpress                             Nginx front-end for the official wordpress:f…   12                                      [OK]
arm32v7/wordpress                                 The WordPress rich content management system…   9
1and1internet/ubuntu-16-nginx-php-5.6-wordpress-4 ubuntu-16-nginx-php-5.6-wordpress-4             7                                       [OK]
```

- Enable Docker Content Trust
  This feature allows you to verify the authenticity, integrity, and publication date of all Docker images available on the Docker Hub Registry. Thing is, Content Trust isn't enabled by default. Once enabled, Docker will be unable to pull down images that have not been signed.



```
avielb@Aviels-MBP [10:31:31] [~/GitHub/k8s-experts/advanced-orchestration] [master]
-> % export DOCKER_CONTENT_TRUST=1
avielb@Aviels-MBP [10:31:46] [~/GitHub/k8s-experts/advanced-orchestration] [master]
-> % docker pull etopian/alpine-php-wordpress
Using default tag: latest
Error: remote trust data does not exist for docker.io/etopian/alpine-php-wordpress: notary.docker.io does not have trust data for docker.io/etopian/alpine-php-wordpress
avielb@Aviels-MBP [10:31:52] [~/GitHub/k8s-experts/advanced-orchestration] [master]
-> %
```

- Set resource limits for your containers
  You can actually set resource limits for your individual containers right from the run command.
- Consider a third-party security tool

- Use Docker Bench Security
  here's a very handy script you can run against your Docker server that will check:
  - Host Configuration
  - Docker Daemon Configuration
  - Docker Daemon Configuration Files
  - Container Images
  - Build Files Container Runtime
- How to run docker bench security script:
  - Open up a terminal window on your Docker server
  - Download the script with the command git clone https://github.com/docker/docker-bench-security.git
  - Change into the newly created directory with the command cd docker-bench-security
  - Run the script with the command sudo sh docker-bench-security.sh
- How to run kube bench:
  - git clone https://github.com/aquasecurity/kube-bench.git
  - cd kube-bench, run kubectl apply -f job.yaml
  - Wait for the job to finish and than view it’s logs with kubectl logs kube-bench-123

```
$ kubectl apply -f job.yaml
job.batch/kube-bench created

$ kubectl get pods
NAME                      READY   STATUS              RESTARTS   AGE
kube-bench-j76s9   0/1     ContainerCreating   0          3s

# Wait for a few seconds for the job to complete
$ kubectl get pods
NAME                      READY   STATUS      RESTARTS   AGE
kube-bench-j76s9   0/1     Completed   0          11s

# The results are held in the pod's logs
kubectl logs kube-bench-j76s9
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
...
```

- As Kubernetes is entirely API driven, controlling and limiting who can access the cluster and what actions they are allowed to perform is the first line of defense.
- Use Transport Level Security (TLS) for all API traffic - Kubernetes expects that all API communication in the cluster is encrypted by default with TLS, and the majority of installation methods will allow the necessary certificates to be created and distributed to the cluster components.
- Choose an authentication mechanism for the API servers to use that matches the common access patterns when you install a cluster.
- For instance, small single user clusters may wish to use a simple certificate or static Bearer token approach.

Kubernetes Infrastructure Security

- A kubeconfig file is a file used to configure access to Kubernetes when used in conjunction with the kubectl command line tool (or other clients).
- This kubeconfig file and its contents are specific to the cluster you are viewing. You will need a separate kubeconfig file for each cluster that you have access to.
- Use kubeconfig files to organize information about clusters, users, namespaces, and authentication mechanisms.
- The kubectl command-line tool uses kubeconfig files to find the information it needs to choose a cluster and communicate with the API server of a cluster.
- The kubeconfig file is built of the definition of the following objects:
- Cluster - the API server address and SSL/TLS certificate used to connect the kubernetes API server.
- Context - A context element in a kubeconfig file is used to group access parameters under a convenient name. Each context has three parameters: cluster, namespace, and user.
- By default, the kubectl command-line tool uses parameters from the current context to communicate with the cluster.
- User - A user we will use to authenticate to our cluster

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /Users/avielb/.minikube/ca.crt
    server: https://192.168.99.101:8443
  name: minikube
contexts:
- context:
    cluster: minikube
    user: minikube
  name: minikube
current-context: minikube
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /Users/avielb/.minikube/client.crt
    client-key: /Users/avielb/.minikube/client.key
```

- Kubernetes authorizes API requests using the API server.
- It evaluates all of the request attributes against all policies and allows or denies the request.
- All parts of an API request must be allowed by some policy in order to proceed. This means that permissions are denied by default.

Kubernetes allows several authorization modules:

- Node - A special-purpose authorizer that grants permissions to kubelets based on the pods they are scheduled to run.
- ABAC - Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc).
- RBAC - Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file.

Kubernetes Application Security

- Kubernetes secret objects let you store and manage sensitive information, such as passwords, OAuth tokens, and ssh keys.
- Putting this information in a secret is safer and more flexible than putting it verbatim in a Pod Lifecycle definition or in a container image
- To use a secret, a pod needs to reference the secret.
- A secret can be used with a pod in two ways:
    - As files in a volume mounted on one or more of its containers
    - Used by kubelet when pulling images for the pod.

Creating secrets by commands:

1. echo -n 'admin' > ./username.txt
2. echo -n '1f2d1e2e67df' > ./password.txt
3. kubectl create secret generic db-user-pass --from-file=./username.txt --from-file=./password.txt
4. kubectl get secrets
5. kubectl describe secrets/db-user-pass

Pod Security Policy

- A Pod Security Policy is a cluster-level resource that controls security sensitive aspects of the pod specification.
- The PodSecurityPolicy objects define a set of conditions that a pod must run with in order to be accepted into the system, as well as defaults for the related fields.

Control Aspect	Field Names
Running of privileged containers	`privileged`
Usage of host namespaces	`hostPID`, `hostIPC`
Usage of host networking and ports	`hostNetwork`, `hostPorts`
Usage of volume types	`volumes`
Usage of the host filesystem	`allowedHostPaths`
White list of Flexvolume drivers	`allowedFlexVolumes`
Allocating an FSGroup that owns the pod's volumes	`fsGroup`
Requiring the use of a read only root file system	`readOnlyRootFilesystem`
The user and group IDs of the container	`runAsUser`, `runAsGroup`, `supplementalGroups`
Restricting escalation to root privileges	`allowPrivilegeEscalation`, `defaultAllowPrivilegeEscalation`
Linux capabilities	`defaultAddCapabilities`, `requiredDropCapabilities`, `allowedCapabilities`
The SELinux context of the container	`seLinux`

Kubernetes Advanced Orchestration

Kubernetes ReplicaSet

- A ReplicaSet's purpose is to maintain a stable set of replica Pods running at any given time.
- As such, it is often used to guarantee the availability of a specified number of identical Pods.
- A ReplicaSet is defined with fields, including a selector that specifies how to identify Pods it can acquire, a number of replicas indicating how many Pods it should be maintaining, and a pod template specifying the data of new Pods it should create to meet the number of replicas criteria.
- A ReplicaSet then fulfills its purpose by creating and deleting Pods as needed to reach the desired number.
- A ReplicaSet ensures that a specified number of pod replicas are running at any given time.
- However, a Deployment is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.
- Therefore, it is recommended using Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don't require updates at all.

Kubernetes StatefulSet

- StatefulSet is the workload API object used to manage stateful applications.
- Manages the deployment and scaling of a set of Pods , and provides guarantees about the ordering and uniqueness of these Pods.
- StatefulSets are valuable for applications that require one or more of the following:
    - Stable - unique network identifiers.
    - Stable - persistent storage.
    - Ordered - graceful deployment and scaling.
    - Ordered - automated rolling updates.

Kubernetes DaemonSet

- A DaemonSet ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them.
- As nodes are removed from the cluster, those Pods are garbage collected.
- Deleting a DaemonSet will clean up the Pods it created.
- Some typical uses of a DaemonSet are:
    - running a cluster storage daemon, such as glusterd, ceph, on each node.
    - running a logs collection daemon on every node, such as fluentd or logstash.
    - running a node monitoring daemon on every node, such as Prometheus Node Exporter, Sysdig Agent, collectd, Dynatrace OneAgent, AppDynamics Agent, Datadog agent, New Relic agent, Ganglia gmond or Instana agent.
- In a simple case, one DaemonSet, covering all nodes, would be used for each type of daemon.
- A more complex setup might use multiple DaemonSets for a single type of daemon, but with different flags and/or different memory and cpu requests for different hardware types.

Kubernetes Cron Jobs

- You can use Cron Jobs to run jobs on a time-based schedule.
- These automated jobs run like Cron tasks on a Linux or UNIX system.
- Cron jobs are useful for creating periodic and recurring tasks, like running backups or sending emails.
- Cron jobs can also schedule individual tasks for a specific time, such as if you want to schedule a job for a low activity period.