Core

Pods

- A Pod represents a collection of application containers and volumes running in the same execution environment.
- Pods are the smallest deployable artifact in a Kubernetes cluster.

Reference : https://kubernetes.io/docs/concepts/workloads/pods/

Imperative Using Cli

- Schedule an aspnet container using kubectl `kubectl run aspnet-cli --image mcr.microsoft.com/dotnet/samples:aspnetapp`
- Verify the pod is running `kubectl get pods`
- Expose a port on your local machine to connect to the pod

  ```
  # a secure tunnel is created from your local machine, through the Kubernetes
  master, to the instance of the Pod running on one of the worker nodes
  kubectl port-forward aspnet-cli 8080:80
  ```

Browse to `http://localhost:8080/` to validate.

Declarative Using YAML - preferred

~~~
``` powershell
 kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/c24971c6d92dd2c2d97f000d2e5e87cca379fddd/k8sws-core-pods.yaml

 # Expose a port
 kubectl port-forward k8sws-core-pods 8081:80

 # Delete
 kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/c24971c6d92dd2c2d97f000d2e5e87cca379fddd/k8sws-core-pods.yaml
```
~~~

ReplicaSet

Reference : https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

The limitation with pods alone is that they can be terminated either by the system to free up resources or by an user. Once a pod dies, it does not come up automatically. More often than not, we want multiple replicas of a container running at a particular time for a variety of reasons such as redundancy (tolerate failure) and scalability (handle more requests).

- A ReplicaSet ensures that the right types and number of Pods are running at all times.
- ReplicaSets are the building blocks used to describe common application deployment patterns and provide the underpinnings of self-healing for our applications at the infrastructure level.
- Pods managed by ReplicaSets are automatically rescheduled under certain failure conditions, such as node failures and network partitions
- A replicaset is linked to pods using labels and label selectors. In the example below, the pods are labelled with `app: k8sws-core-replicasets` and this is used by the replicaet to identoty the pods. Its possible and sometimes necessary to have more than one label for identification.

  ```
   kubectl apply -f
  https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54c
  d918/raw/1591d46e1578082ff6313edfae423bfc55aeeddd/k8sws-core-
  replicasets.yaml

   kubectl scale rs k8sws-core-replicasets --replicas=4
   # Expose a port
   kubectl port-forward rs/k8sws-core-replicasets 8081:80
  ```

```
 kubectl delete pod XXX

 # Delete
 kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54c
d918/raw/1591d46e1578082ff6313edfae423bfc55aeeddd/k8sws-core-
replicasets.yaml
```

Pros : First step towards desired state. Automatic failover Cons : Only one replicasets

Deployments

Reference : https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Both Pods and ReplicaSets are expected to be tied to specific container images that don't change i.e. they are used to build a single instance of your application but they do little to help you manage the daily or weekly deployment of releasing new versions of your application.

- The Deployment object exists to manage the release of new versions. Deployments represent deployed applications in a way that transcends any particular version. Additionally, deployments enable you to easily move from one version of your code to the next. This "rollout" process is specifiable and careful.
- In all likelihood, a deployment will be the unit of deployment for your service i.e. for each of your microservice or container, you will have a deployment object which will internally create replicasets and pods and manage those automatically.
- Similar to replicasets, a deployment is linked to pods using labels and selectors.

```
 kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54c
d918/raw/8039987b3c8a07142f989f06f73b750a81091526/k8sws-core-
deployments.yaml

 kubectl get deploy,rs,pods

 kubectl scale deploy k8sws-core-deployments --replicas=4
 #delete a pod
 kubectl delete pod XXX

 # Expose a port
 kubectl port-forward deploy/k8sws-core-deployments 8081:80

 # new tab
 while($true)
    {
    kubectl get pods
    Start-Sleep -Seconds 5

    }

 #Update the pod's image
 kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54c
d918/raw/61a29ff9ac4b2c455720dc9101bf2b674c146399/k8sws-core-deployments-
v2.yaml

# get history of deployments
 kubectl rollout history deploy/k8sws-core-deployments
# revert to previous deployment
kubectl rollout undo deploy/k8sws-core-deployments --to-revision=1
# test revert
kubectl port-forward deploy/k8sws-core-deployments 8081:80

 # Delete
 kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54c
d918/raw/8039987b3c8a07142f989f06f73b750a81091526/k8sws-core-
deployments.yaml

 kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54c
d918/raw/61a29ff9ac4b2c455720dc9101bf2b674c146399/k8sws-core-deployments-
v2.yaml
```

Services

Reference : https://kubernetes.io/docs/concepts/services-networking/service/

The best explanation for the motivation and reasoning behind services is perhaps best explained in the kubernetes documentation itself which is quoted below.

> Kubernetes Pods are created and destroyed to match the state of your cluster. Pods are nonpermanent resources. If you use a Deployment to run your app, it can create and destroy Pods dynamically.
>
> Each Pod gets its own IP address, however in a Deployment, the set of Pods running in one moment in time could be different from the set of Pods running that application a moment later.
>
> This leads to a problem: if some set of Pods (call them "backends") provides functionality to other Pods (call them "frontends") inside your cluster, how do the frontends find out and keep track of which IP address to connect to, so that the frontend can use the backend part of the workload?
>
> Enter Services.

Note: Similar to deployments, services are also linked to pods via labels and selectors

Service Types

ClusterIP

Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType

~~~
``` powershell
kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/1ceccb53198c308549ebb3ad73cb6fd978938d98/k8sws-core-services-clusterip.yaml

#verify
kubectl get svc,deploy
#ips match
kubectl get pods -o wide
kubectl describe svc k8sws-core-services-clusterip
#go in detail

```
~~~

Because the service is a ClusterIP (i.e internal service), it can only be accessed by callers that are inside the kubernetes network. To test that the service is actually running, install a test image

```
kubectl run nginx-standalone --image nginx
# log into  the newly created pod
kubectl exec -it nginx-standalone -- bash
# ip address of the service created - will differ per workstation
curl http://10.107.199.116
```

NodePort

Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You'll be able to contact the NodePort Service, from outside the cluster, by requesting :.

NodePort is what you'd typically use for local debugging.

~~~
``` powershell
kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/9d2b0b6dfc31736bc6eb3d8063a398f6a730dba5/k8sws-core-services-nodeport.yaml

#verify
kubectl get svc
```
~~~

> The remaining services types are when deploying to a k8s cluster and we will cover these in a later exercise.

LoadBalancer

Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created

ExternalName

Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

Ingress

Reference : https://kubernetes.io/docs/concepts/services-networking/ingress/ While Ingress is not exactly a kubernetes service type, it does have similarities in functionality to the Load Balancer service type. If most of the communication happens over http, an ingress (which doe operate at a Layer 7 - http - level) offers many advantages among which are a single point of entry to your cluster and routing internally via host names and path mappings. As part of a later exercise, we will deploy an ingress resource.

DNS in services

Similar to docker compose, kubernetes has built in dns - what that means is that if you are calling a service from a pod...you can use the dns name of that service (the service name in the yaml) and that does not need to change even when you are deploying to another cluster (environment promotion). Lets take a look at this

~~~
``` powershell
#service B
kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svcb.yaml
# service A
kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svca.yaml

#verify
kubectl get svc
#http://localhost:XXX
#http://localhost:XXX
#
#chain call
#http://localhost:XXXX/chaincall/chaincall?
delayInSeconds=2&propagate=false&forwardHeaders=true

#cleanup
kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
~~~

```
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svcb.yaml
kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svca.yaml

```
```

Labels, Selectors and Annotations

Kubernetes was made to grow with you as your application scales in both size and complexity. With this in mind, labels and annotations were added as foundational concepts. Labels and annotations let you work in sets of things that map to how you think about your application. You can organize, mark, and cross-index all of your resources to represent the groups that make the most sense for your application.

Labels are key/value pairs that can be attached to Kubernetes objects such as Pods and ReplicaSets. They can be arbitrary, and are useful for attaching identifying information to Kubernetes objects. Labels provide the foundation for grouping objects.

Annotations, on the other hand, provide a storage mechanism that resembles labels: annotations are key/value pairs designed to hold nonidentifying information that can be leveraged by tools and libraries.

There is overlap, and there is no right or wrong. The way i think of it....labels are for humans to read and annotations for the most part are for machines to read. When in doubt, add information to an object as an annotation and promote it to a label if you find yourself wanting to use it in a selector.

````
``` powershell
kubectl get pods --show-labels
# select - can apply to deployment resource too
kubectl get pods --selector="app=k8sws-core-deployments-nodeport"
kubectl get pods --selector="app in (k8sws-core-deployments-nodeport,k8sws-core-
deployments)"
#is a label even set at all
kubectl get pods --selector="app"
#is a label NOT set at all
kubectl get pods --selector="!app"

```
````

> From K8s up and running Page 71 or 92 In addition to enabling users to organize their infrastructure, labels play a critical role in linking various related Kubernetes objects. Kubernetes is a purposefully decoupled system. There is no hierarchy and all components operate independently. However, in many cases objects need to relate to one another, and these relationships are defined by labels and label selectors. For example, ReplicaSets, which create and maintain multiple replicas of a Pod, find the Pods that they are managing via a selector. Likewise, a service load balancer finds the Pods it should bring traffic to via a selector query. When a Pod is created, it can use a node selector to identify a particular set of nodes that it can be scheduled onto. When people want to restrict network traffic in their cluster, they use NetworkPolicy in conjunction with specific labels to identify Pods that should or should not be
````

> allowed to communicate with each other. Labels are a powerful and ubiquitous glue that holds a Kubernetes application together. Though your application will likely start out with a simple set of labels and queries, you should expect it to grow in size and sophistication with time.

> Annotations are used in various places in Kubernetes, with the primary use case being rolling deployments. During rolling deployments, annotations are used to track rollout status and provide the necessary information required to roll back a deployment to a previous state.

## Configuration - ConfigMaps and Secrets

Reference : https://kubernetes.io/docs/concepts/configuration/configmap/ Reference : https://kubernetes.io/docs/concepts/configuration/secret/

> Config maps and Secrets are two standard ways to store and inject configuration data into pods. The primary difference between the two is that ConfigMaps store data in plain text whereas secrets store data as base64 encoded - easily decryptable. A general rule of thumb would be that use secrets if you app is a backend app where you are storing passwords etc but use config maps for front end configuration because that is public anyway. The more fundamental difference, in my opinion is if you are teh admin and you have this segregation between configmaps and secrets you can do RBAC ...so allow someone to read configmaps but not secrets

- Via CLI

```
#SECRET
kubectl create secret generic aspnetapp-secret-cli --from-literal=VersionPathBase=HelloFromSecret
# verify the secret was created - notice that the data column shows how many data does the secret store
kubectl get secret/aspnetapp-secret-cli
# view a description of the Secret
kubectl describe secret/aspnetapp-secret-cli
# Decoding the Secret - get encoded
kubectl get secret/aspnetapp-secret-cli -o jsonpath='{.data}'
# decode
[Text.Encoding]::Utf8.GetString([Convert]::FromBase64String('SGVsbG9Gcm9tU2VjcmV0'))
```

- Via YAML

```
kubectl apply -f https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/raw/9effb81f19a0598343fb28e0e75d3762e0f12175/k8sws-core-configs-secrets-cm.yaml

# get services
kubectl get svc
# browse to http://localhost:XXX/metrics
```

> In the above example we are showing two methods of loading configmaps and secrets. 1. Loading certain configurations entries individually from specific properties in either cm or secrets. 2. Loading all the keys in a secret as environment variables to a container - in fact this is the pattern that we use pretty commonly where we define 1 cm or secret resource per microservice so that data updated in one secret file does not unintentionally impact other services.

There is another pattern for loading secrets and config maps where we load the entries in the file system. The difference here is that the application has to be aware of the mounted file and be able to load from there.

```
#create a config map from file
kubectl create cm file-cm--appsettings --from-file=./appsettings.configmaps.json
#verify
kubectl describe cm file-cm--appsettings
#create the deployment and service
kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/raw/3302afa4360e46666372cdc4457c598fb54a0b83/k8sws-core-deployments-configs-file-cm.yaml
#test
# http://localhost:XXX/chaincallc/filemountedconfig?configKey=RandomKey
# Update the configs
# Since there is no kubectl update
kubectl create cm file-cm--appsettings --from-file=./appsettings.configmaps.json --dry-run=client -o yaml | kubectl apply -f -
 #2 remember anything mounted inside a container is at that.....so delete pods
 kubectl delete pods --all

 #cleanup
 kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/raw/3302afa4360e46666372cdc4457c598fb54a0b83/k8sws-core-deployments-configs-file-cm.yaml

 kubectl delete cm --all
```

## Namespaces

Reference : https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

> Namespaces are used to support multiple virtual clusters backed by the same physical cluster. The virtual clusters may be per environment (dev, qa etc) or by workload type (appdev, bi) or even by technology stack (frontend, backend).

From an administrator's perspective, we can then assign roles and role binding per namespace and segregate one bad actor from bringing down the entire system. Furthermore we can assign resource quotas per namespace so that all the resources combined in a namespace do not exceed the configured quota.

**Example 1**

```
# get all namespaces
kubectl get namespace

# create new namespace
kubectl create namespace demo-ns
# setting the namespace for a request - adhoc basis i.e. if you want to run this
one command under another namespace
kubectl run nginx --image=nginx --namespace=demo-ns
#getting pods under the default namespace
kubectl get pods # should not return the pod that we just created
# specifying the namespace should return that pod
kubectl get pods --namespace=demo-ns

# permanently save the namespace for all subsequent kubectl commands in that
context
kubectl config set-context --current --namespace=demo-ns
 #getting pods under the default namespace which we just switched to demo-ns
kubectl get pods # should now return our pods

# switch back to default namespace
kubectl config set-context --current --namespace=default

# delete a namespace - will delete all resources created under that namespace
kubectl delete namespaces demo-ns
```

Lets take a look at a more realistic scenario. Lets go back to an earlier scenario of Service A calling Service B. We had established that service A can call service B since they were both in the same namespace. In this example we will look at deploying service B to another namespace and redo that test.

```
  kubectl delete deploy --all
  kubectl delete svc --all
  kubectl delete secret --all
  kubectl delete cm --all

  #service B
  kubectl create namespace demo-ns

  kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svcb.yaml -n demo-
ns
  # service A
  kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svca.yaml

  #verify
  kubectl get svc
  #THIS SHOULD FAIL
  #http://localhost:XXXX/chaincall/chaincall?
delayInSeconds=2&propagate=false&forwardHeaders=true

  #LETS FIX THIS
  kubectl edit deploy/k8sws-core-deployments-namespace-svca
  #update url to http://k8sws-core-services-namespace-svcb.demo-
ns.svc.cluster.local
  #format is servicename.namespace.svc.cluster.local
  #DELETE PODS so that env vars are reloaded
  kubectl delete pods --all
  #testing now should work

  #cleanup
  kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svcb.yaml -n demo-
ns
  kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/983ddaea3e253e45c80a7f61aa9ac945ea91836d/k8sws-core-services-svca.yaml

  kubectl delete namespaces demo-ns

  #browse to http://localhost:31901/chaincall/chaincall?
delayInSeconds=2&propagate=false&forwardHeaders=true
```

## Health Checks

Reference : https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

**Liveness Probe**

Many applications running for long periods of time eventually transition to broken states, and cannot recover except by being restarted. Kubernetes provides liveness probes to detect and remedy such situations.

**Readiness Probe**

Sometimes, applications are temporarily unable to serve traffic. For example, an application might need to load large data or configuration files during startup, or depend on external services after startup. In such cases, you don't want to kill the application, but you don't want to send it requests either. Kubernetes provides readiness probes to detect and mitigate these situations. A pod with containers reporting that they are not ready does not receive traffic through Kubernetes Services.

```
    kubectl delete deploy --all
    kubectl delete svc --all
    kubectl delete secret --all
    kubectl delete cm --all

    kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/raw/2071c7e0e483a6b7c769828b06d4a3c0140df9ab/k8sws-core-deploy-probes.yaml

    # verify
    # http://localhost:XX/admin/live
    # http://localhost:XX/admin/ready

    # lets kill the live probe
    #http://localhost:30133/admin/live/mutate
    # check again
    # http://localhost:30133/admin/live -- should be back up and running
    # notice that the restart column increments

    # lets kill the readiness probe
    #http://localhost:30133/admin/ready/mutate
    # check again
    # http://localhost:30133/admin/ready -- DOES NOT AUTO RESTART. get back a 429
    # notice that the ready column shows NOT READY. THIS WILL NOT AUTO HEAL unless
the app changes


    #cleanup
    kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/raw/2071c7e0e483a6b7c769828b06d4a3c0140df9ab/k8sws-core-deploy-probes.yaml
```

## Container Resources

Reference : https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

When you specify a pod, as a best practice you want to specify how much of each resource (cpu and memory) the container needs. There are two aspects to this

1. Request : When you specify the resource **request** for Containers in a Pod, the scheduler uses this information to decide which node to place the Pod on.The kubelet also reserves at least the request amount of that system resource specifically for that container to use.
2. Limit: When you specify a resource **limit** for a Container, the kubelet enforces those limits so that the running container is not allowed to use more of that resource than the limit you set.

> If the node where a Pod is running has enough of a resource available, it's possible (and allowed) for a container to use more resource than its request for that resource specifies. However, a container is not allowed to use more than its resource limit.
> For example, if you set a memory request of 256 MiB for a container, and that container is in a Pod scheduled to a Node with 8GiB of memory and no other Pods, then the container can try to use more RAM.

> If you set a memory limit of 4GiB for that Container, the kubelet (and container runtime) enforce the limit. The runtime prevents the container from using more than the configured resource limit. For example: when a process in the container tries to consume more than the allowed amount of memory, the system kernel terminates the process that attempted the allocation, with an out of memory (OOM) error.
> If a Container specifies its own memory limit, but does not specify a memory request, Kubernetes automatically assigns a memory request that matches the limit. Similarly, if a Container specifies its own CPU limit, but does not specify a CPU request, Kubernetes automatically assigns a CPU request that matches the limit

- CPU: One cpu, in Kubernetes, is equivalent to 1 vCPU/Core for cloud providers.
- Memory: Limits and requests for memory are measured in bytes

```
    kubectl delete deploy --all
    kubectl delete svc --all
    kubectl delete secret --all
    kubectl delete cm --all

    #no resource yaml
    kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/054644acc1db8d22b076577e565cf1eb863e46a0/k8sws-core-deployments-svcc-
noresource.yaml

    # verify
    # http://localhost:30113/admin/live

    #get container stats
    docker container ls
    docker stats d40f679691b8

    # notice how refreshing http://localhost:30113/admin/live does not change
value much
    # try this http://localhost:30113/admin/memorySpike
    # notice now the memory spike - so in this case estimate for memory is 300MB
    # try this http://localhost:30113/admin/cpuSpike?limitInSeconds=10
    # notice the cpu spike --> so i might say my max cpu is 1 vcpu

    # lets apply the resources and deploy again
    kubectl apply -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/054644acc1db8d22b076577e565cf1eb863e46a0/k8sws-core-deployments-svcc.yaml
    #delete pods
    kubectl delete pods --all

    #cleanup
    kubectl delete -f
https://gist.githubusercontent.com/nishantnepal/fcae1e0c599f75bdb977f555e54cd918/r
aw/054644acc1db8d22b076577e565cf1eb863e46a0/k8sws-core-deployments-svcc.yaml
```

#### BONUS - What happens if your process wants more resource than limit

```
# lets edit the deployment to limit memory
kubectl edit deploy/k8sws-core-deployments-svcc
# update the memory limit from 500 MB to 250 MB and cpu from I cpu to 500m
# delete the pods
kubectl delete pods --all
#run the same request again http://localhost:30113/admin/memorySpike  - this time
it errors out
# .net core i.e the process running inside the container sees only a certain
memory
kubectl logs deploy/k8sws-core-deployments-svcc
```

> In day-to-day operation, this means that in case of overcommitting resources, pods without limits will likely be killed, containers using more resources than requested have some chances to die and guaranteed containers will most likely be fine.
>
> There is a great difference between CPU and memory quota management. Regarding memory, a pod without requests and limits is considered burstable and is the first of the list to OOM kill. With the CPU, this is not the case. A pod without CPU limits is free to use all the CPU resources in the node. Well, truth is, the CPU is there to be used, but if you can’t control which process is using your resources, you can end up with a lot of problems due to CPU starvation of key processes.