**KUBERNETES**

$ minikube version

minikube version: v1.3.0

commit: 43969594266d77b555a207b0f3e9b3fa1dc92b1f

$ minikube start --wait=false

\* minikube v1.3.0 on Ubuntu 18.04

\* Running on localhost (CPUs=2, Memory=2461MB, Disk=47990MB) ...

\* OS release is Ubuntu 18.04.2 LTS

\* Preparing Kubernetes v1.15.0 on Docker 18.09.5 ...

- kubelet.resolv-conf=/run/systemd/resolve/resolv.conf

\* Pulling images ...

\* Launching Kubernetes ...

\* Done! kubectl is now configured to use "minikube"

I1031 10:50:28.852169 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.852428 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.872652 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.874304 1 client.go:354] parsed scheme: ""

I1031 10:50:28.874316 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.874452 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.874509 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.885375 1 client.go:354] parsed scheme: ""

I1031 10:50:28.885397 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.885435 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.885543 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.885934 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.897481 1 client.go:354] parsed scheme: ""

I1031 10:50:28.897577 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.897779 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.897961 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.898308 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.906258 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.906855 1 client.go:354] parsed scheme: ""

I1031 10:50:28.906867 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.906971 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.907075 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.916171 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.917348 1 client.go:354] parsed scheme: ""

I1031 10:50:28.917485 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.917605 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.917765 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.932010 1 client.go:354] parsed scheme: ""

I1031 10:50:28.932216 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.932219 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.932370 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.932503 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.941897 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.943007 1 client.go:354] parsed scheme: ""

I1031 10:50:28.943089 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.943186 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.943274 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.952733 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.953474 1 client.go:354] parsed scheme: ""

I1031 10:50:28.953607 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.953718 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.953909 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.962498 1 client.go:354] parsed scheme: ""

I1031 10:50:28.962981 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.962764 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.963172 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.963297 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.970931 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.971519 1 client.go:354] parsed scheme: ""

I1031 10:50:28.971530 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.971572 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.971743 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.995862 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:28.997084 1 client.go:354] parsed scheme: ""

I1031 10:50:28.997205 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:28.997346 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:28.997495 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.010371 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.011227 1 client.go:354] parsed scheme: ""

I1031 10:50:29.011245 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.011394 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.011482 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.025441 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.026291 1 client.go:354] parsed scheme: ""

I1031 10:50:29.026307 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.026418 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.026545 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.040487 1 client.go:354] parsed scheme: ""

I1031 10:50:29.040975 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.040772 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.041221 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.041405 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.051884 1 client.go:354] parsed scheme: ""

I1031 10:50:29.052124 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.052250 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.052097 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.052624 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.063453 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.064296 1 client.go:354] parsed scheme: ""

I1031 10:50:29.064323 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.064353 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.064423 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.077423 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.079490 1 client.go:354] parsed scheme: ""

I1031 10:50:29.079713 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.079836 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.079992 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.090643 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.091412 1 client.go:354] parsed scheme: ""

I1031 10:50:29.091436 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.091606 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.091690 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.103063 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.104076 1 client.go:354] parsed scheme: ""

I1031 10:50:29.104147 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.104260 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.104422 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.117890 1 client.go:354] parsed scheme: ""

I1031 10:50:29.118049 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.118124 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.118250 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.118507 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.130917 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.132012 1 client.go:354] parsed scheme: ""

I1031 10:50:29.132112 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.132263 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.132390 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.142838 1 client.go:354] parsed scheme: ""

I1031 10:50:29.143392 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.143441 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.143098 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.143656 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.154027 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.155782 1 client.go:354] parsed scheme: ""

I1031 10:50:29.155797 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.155962 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.156071 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.165428 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.166305 1 client.go:354] parsed scheme: ""

I1031 10:50:29.166364 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:29.166458 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:29.166640 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:29.189776 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

W1031 10:50:29.329974 1 genericapiserver.go:351] Skipping API batch/v2alpha1 because it has no resources.

W1031 10:50:29.348661 1 genericapiserver.go:351] Skipping API node.k8s.io/v1alpha1 because it has no resources.

W1031 10:50:29.362395 1 genericapiserver.go:351] Skipping API rbac.authorization.k8s.io/v1alpha1 because it has no resources.

W1031 10:50:29.364632 1 genericapiserver.go:351] Skipping API scheduling.k8s.io/v1alpha1 because it has no resources.

W1031 10:50:29.371020 1 genericapiserver.go:351] Skipping API storage.k8s.io/v1alpha1 because it has no resources.

E1031 10:50:30.218965 1 prometheus.go:55] failed to register depth metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219016 1 prometheus.go:68] failed to register adds metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219050 1 prometheus.go:82] failed to register latency metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219078 1 prometheus.go:96] failed to register workDuration metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219105 1 prometheus.go:112] failed to register unfinished metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219127 1 prometheus.go:126] failed to register unfinished metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219147 1 prometheus.go:152] failed to register depth metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219165 1 prometheus.go:164] failed to register adds metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219207 1 prometheus.go:176] failed to register latency metric admission\_quota\_controller: duplicate metrics collector registration attempted

E1031 10:50:30.219251 1 prometheus.go:188] failed to register work\_duration metric admission\_quota\_controller: duplicate metrics collector registration

attempted

E1031 10:50:30.219282 1 prometheus.go:203] failed to register unfinished\_work\_seconds metric admission\_quota\_controller: duplicate metrics collector

registration attempted

E1031 10:50:30.219304 1 prometheus.go:216] failed to register longest\_running\_processor\_microseconds metric admission\_quota\_controller: duplicate metrics

collector registration attempted

I1031 10:50:30.219326 1 plugins.go:158] Loaded 10 mutating admission controller(s) successfully inthe following order:

NamespaceLifecycle,LimitRanger,ServiceAccount,NodeRestriction,TaintNodesByCondition,Priority,DefaultTolerationSeconds,DefaultStorageClass,StorageObjectInUs

eProtection,MutatingAdmissionWebhook.

I1031 10:50:30.219335 1 plugins.go:161] Loaded 6 validating admission controller(s) successfully in the following order:

LimitRanger,ServiceAccount,Priority,PersistentVolumeClaimResize,ValidatingAdmissionWebhook,ResourceQuota.

I1031 10:50:30.221806 1 client.go:354] parsed scheme: ""

I1031 10:50:30.221823 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:30.221861 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:30.221904 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:30.231819 1 client.go:354] parsed scheme: ""

I1031 10:50:30.231843 1 client.go:354] scheme "" not registered, fallback to default scheme

I1031 10:50:30.231889 1 asm\_amd64.s:1337] ccResolverWrapper: sending new addresses to cc: [{127.0.0.1:2379 0 <nil>}]

I1031 10:50:30.231959 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:30.232240 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:30.246102 1 asm\_amd64.s:1337] balancerWrapper: got update addr from Notify: [{127.0.0.1:2379 <nil>}]

I1031 10:50:32.114168 1 secure\_serving.go:116] Serving securely on [::]:8443

I1031 10:50:32.114472 1 crd\_finalizer.go:255] Starting CRDFinalizer

I1031 10:50:32.119067 1 autoregister\_controller.go:140] Starting autoregister controller

I1031 10:50:32.119289 1 cache.go:32] Waiting for caches to sync for autoregister controller

I1031 10:50:32.121518 1 available\_controller.go:374] Starting AvailableConditionController

I1031 10:50:32.121645 1 cache.go:32] Waiting for caches to sync for AvailableConditionController controller

I1031 10:50:32.124689 1 apiservice\_controller.go:94] Starting APIServiceRegistrationController

I1031 10:50:32.124838 1 cache.go:32] Waiting for caches to sync for APIServiceRegistrationController controller

I1031 10:50:32.124925 1 controller.go:81] Starting OpenAPI AggregationController

I1031 10:50:32.125060 1 crdregistration\_controller.go:112] Starting crd-autoregister controller

I1031 10:50:32.125140 1 controller\_utils.go:1029] Waiting for caches to sync for crd-autoregister controller

I1031 10:50:32.119740 1 controller.go:83] Starting OpenAPI controller

I1031 10:50:32.119750 1 customresource\_discovery\_controller.go:208] Starting DiscoveryController

I1031 10:50:32.119776 1 naming\_controller.go:288] Starting NamingConditionController

I1031 10:50:32.119782 1 establishing\_controller.go:73] Starting EstablishingController

I1031 10:50:32.119786 1 nonstructuralschema\_controller.go:191] Starting NonStructuralSchemaConditionController

E1031 10:50:32.134805 1 controller.go:148] Unable to remove old endpoints from kubernetes service:StorageError: key not found, Code: 1, Key:

/registry/masterleases/172.17.0.26, ResourceVersion: 0, AdditionalErrorMsg:

I1031 10:50:32.237422 1 cache.go:39] Caches are synced for autoregister controller

I1031 10:50:32.238067 1 cache.go:39] Caches are synced for AvailableConditionController controller

I1031 10:50:32.238217 1 cache.go:39] Caches are synced for APIServiceRegistrationController controller

I1031 10:50:32.238357 1 controller\_utils.go:1036] Caches are synced for crd-autoregister controller

I1031 10:50:33.111744 1 controller.go:107] OpenAPI AggregationController: Processing item

I1031 10:50:33.111932 1 controller.go:130] OpenAPI AggregationController: action for item : Nothing (removed from the queue).

I1031 10:50:33.112008 1 controller.go:130] OpenAPI AggregationController: action for item k8s\_internal\_local\_delegation\_chain\_0000000000: Nothing (removed

from the queue).

I1031 10:50:33.123360 1 storage\_scheduling.go:119] created PriorityClass system-node-critical withvalue 2000001000

I1031 10:50:33.135009 1 storage\_scheduling.go:119] created PriorityClass system-cluster-critical with value 2000000000

I1031 10:50:33.135048 1 storage\_scheduling.go:128] all system priority classes are created successfully or already exist.

I1031 10:50:34.899747 1 controller.go:606] quota admission added evaluator for: roles.rbac.authorization.k8s.io

I1031 10:50:35.080925 1 controller.go:606] quota admission added evaluator for: endpoints

I1031 10:50:35.180202 1 controller.go:606] quota admission added evaluator for: rolebindings.rbac.authorization.k8s.io

W1031 10:50:35.469834 1 lease.go:223] Resetting endpoints for master service "kubernetes" to [172.17.0.26]

I1031 10:50:35.583684 1 controller.go:606] quota admission added evaluator for: leases.coordination.k8s.io

I1031 10:50:36.590814 1 controller.go:606] quota admission added evaluator for: serviceaccounts

I1031 10:50:36.676504 1 controller.go:606] quota admission added evaluator for: deployments.apps

I1031 10:50:37.044524 1 controller.go:606] quota admission added evaluator for: daemonsets.apps

I1031 10:50:42.715344 1 controller.go:606] quota admission added evaluator for: namespaces

I1031 10:50:43.138085 1 controller.go:606] quota admission added evaluator for: replicasets.apps

I1031 10:50:43.593101 1 controller.go:606] quota admission added evaluator for: controllerrevisions.apps

I1031 10:54:26.042006 1 trace.go:81] Trace[51899172]: "GuaranteedUpdate etcd3: \*v1.Endpoints" (started: 2019-10-31 10:54:25.48784191 +0000 UTC m=

+240.372936615) (total time: 554.121557ms):

Trace[51899172]: [554.075478ms] [533.605882ms] Transaction committed

I1031 10:57:28.983862 1 trace.go:81] Trace[1230320257]: "Get /api/v1/namespaces/kube-system/endpoints/kube-scheduler" (started: 2019-10-31

10:57:28.209615626 +0000 UTC m=+423.094710441) (total time: 774.19858ms):

Trace[1230320257]: [774.131373ms] [774.071327ms] About to write a response

I1031 11:00:03.378366 1 trace.go:81] Trace[261378708]: "List etcd3: key=/cronjobs, resourceVersion=, limit: 500, continue: " (started: 2019-10-31

11:00:02.055346434 +0000 UTC m=+576.940441146) (total time: 1.322980076s):

Trace[261378708]: [1.322980076s] [1.322980076s] END

I1031 11:00:03.378466 1 trace.go:81] Trace[1465900191]: "List /apis/batch/v1beta1/cronjobs" (started: 2019-10-31 11:00:02.055293511 +0000 UTC m=

+576.940388201) (total time: 1.323158309s):

Trace[1465900191]: [1.323088624s] [1.323045516s] Listing from storage done

I1031 11:00:03.404929 1 trace.go:81] Trace[430384980]: "Get /api/v1/namespaces/kube-system/endpoints/kube-controller-manager" (started: 2019-10-31

11:00:02.755386912 +0000 UTC m=+577.640481607) (total time: 629.137556ms):

Trace[430384980]: [629.084143ms] [629.038044ms] About to write a response

I1031 11:06:13.770306 1 trace.go:81] Trace[1705421474]: "Get /api/v1/namespaces/kube-system/endpoints/kube-scheduler" (started: 2019-10-31

11:06:12.412540495 +0000 UTC m=+947.297635194) (total time: 1.357731661s):

Trace[1705421474]: [1.357680526s] [1.357636064s] About to write a response

I1031 11:06:15.168854 1 trace.go:81] Trace[989544190]: "List etcd3: key=/jobs, resourceVersion=, limit: 500, continue: " (started: 2019-10-31 11:06:14.161256555

+0000 UTC m=+949.046351238) (total time: 1.007564819s):

Trace[989544190]: [1.007564819s] [1.007564819s] END

I1031 11:06:15.168942 1 trace.go:81] Trace[476707974]: "List /apis/batch/v1/jobs" (started: 2019-10-31 11:06:14.161206579 +0000 UTC m=+949.046301261) (total

time: 1.007720875s):

Trace[476707974]: [1.007662911s] [1.007621384s] Listing from storage done

I1031 11:06:15.169400 1 trace.go:81] Trace[286723436]: "GuaranteedUpdate etcd3: \*core.Endpoints" (started: 2019-10-31 11:06:13.772670727 +0000 UTC m=

+948.657765421) (total time: 1.396710768s):

Trace[286723436]: [1.396689481s] [1.396511754s] Transaction committed

I1031 11:06:15.169463 1 trace.go:81] Trace[827319983]: "Update /api/v1/namespaces/kube-system/endpoints/kube-scheduler" (started: 2019-10-31

11:06:13.77242482 +0000 UTC m=+948.657519495) (total time: 1.397021287s):

Trace[827319983]: [1.396990238s] [1.39679164s] Object stored in database

I1031 11:06:15.725867 1 trace.go:81] Trace[532669005]: "List etcd3: key=/cronjobs, resourceVersion=, limit: 500, continue: " (started: 2019-10-31

11:06:15.172459505 +0000 UTC m=+950.057554197) (total time: 553.383893ms):

Trace[532669005]: [553.383893ms] [553.383893ms] END

I1031 11:06:15.725920 1 trace.go:81] Trace[337267513]: "List /apis/batch/v1beta1/cronjobs" (started: 2019-10-31 11:06:15.172410472 +0000 UTC m=

+950.057505160) (total time: 553.499718ms):

Trace[337267513]: [553.468508ms] [553.427321ms] Listing from storage done

I1031 11:08:44.732441 1 trace.go:81] Trace[521740563]: "List etcd3: key=/minions, resourceVersion=, limit: 0, continue: " (started: 2019-10-31 11:08:44.01573399

+0000 UTC m=+1098.900828677) (total time: 716.670099ms):

Trace[521740563]: [716.670099ms] [716.670099ms] END

I1031 11:08:44.732989 1 trace.go:81] Trace[1641222334]: "List /api/v1/nodes" (started: 2019-10-31 11:08:44.015714365 +0000 UTC m=+1098.900809051) (total

time: 717.219033ms):

Trace[1641222334]: [716.751388ms] [716.740803ms] Listing from storage done

E1031 11:08:50.126282 1 watcher.go:214] watch chan error: etcdserver: mvcc: required revision has been compacted

==== END logs for container kube-apiserver of pod kube-system/kube-apiserver-minikube ====

==== START logs for container kube-controller-manager of pod kube-system/kube-controller-manager-minikube ====

I1031 10:50:26.463455 1 serving.go:319] Generated self-signed cert in-memory

I1031 10:50:26.826212 1 controllermanager.go:164] Version: v1.15.0

I1031 10:50:26.826810 1 secure\_serving.go:116] Serving securely on 127.0.0.1:10257

I1031 10:50:26.827563 1 deprecated\_insecure\_serving.go:53] Serving insecurely on [::]:10252

I1031 10:50:26.827657 1 leaderelection.go:235] attempting to acquire leader lease kube-system/kube-controller-manager...

E1031 10:50:32.217779 1 leaderelection.go:324] error retrieving resource lock kube-system/kube-controller-manager: endpoints "kube-controller-manager" is

forbidden: User "system:kube-controller-manager" cannot get resource "endpoints" in API group "" in the namespace "kube-system"

I1031 10:50:36.364336 1 leaderelection.go:245] successfully acquired lease kube-system/kube-controller-manager

I1031 10:50:36.364604 1 event.go:258] Event(v1.ObjectReference{Kind:"Endpoints", Namespace:"kube-system", Name:"kube-controller-manager",

UID:"09f8907c-0af2-441a-817b-a9571f5ddbfd", APIVersion:"v1", ResourceVersion:"163", FieldPath:""}): type: 'Normal' reason: 'LeaderElection' minikube\_d0f19b52-

fd8f-4012-a74f-e8dd509c7e1d became leader

I1031 10:50:36.580037 1 plugins.go:103] No cloud provider specified.

I1031 10:50:36.581473 1 controller\_utils.go:1029] Waiting for caches to sync for tokens controller

I1031 10:50:36.683356 1 controller\_utils.go:1036] Caches are synced for tokens controller

I1031 10:50:36.711473 1 controllermanager.go:532] Started "serviceaccount"

W1031 10:50:36.712408 1 controllermanager.go:524] Skipping "root-ca-cert-publisher"

I1031 10:50:36.712580 1 serviceaccounts\_controller.go:117] Starting service account controller

I1031 10:50:36.712676 1 controller\_utils.go:1029] Waiting for caches to sync for service account controller

I1031 10:50:36.812482 1 controllermanager.go:532] Started "namespace"

I1031 10:50:36.812814 1 namespace\_controller.go:186] Starting namespace controller

I1031 10:50:36.812875 1 controller\_utils.go:1029] Waiting for caches to sync for namespace controller

I1031 10:50:36.859593 1 node\_lifecycle\_controller.go:290] Sending events to api server.

I1031 10:50:36.859998 1 node\_lifecycle\_controller.go:323] Controller is using taint based evictions.

I1031 10:50:36.860088 1 taint\_manager.go:175] Sending events to api server.

I1031 10:50:36.860379 1 node\_lifecycle\_controller.go:388] Controller will reconcile labels.

I1031 10:50:36.860441 1 node\_lifecycle\_controller.go:401] Controller will taint node by condition.

I1031 10:50:36.860500 1 controllermanager.go:532] Started "nodelifecycle"

I1031 10:50:36.860550 1 node\_lifecycle\_controller.go:425] Starting node controller

I1031 10:50:36.860617 1 controller\_utils.go:1029] Waiting for caches to sync for taint controller

I1031 10:50:36.887916 1 controllermanager.go:532] Started "persistentvolume-expander"

I1031 10:50:36.887995 1 expand\_controller.go:300] Starting expand controller

I1031 10:50:36.888707 1 controller\_utils.go:1029] Waiting for caches to sync for expand controller

I1031 10:50:37.109449 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor replicasets.extensions

I1031 10:50:37.109567 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor cronjobs.batch

I1031 10:50:37.109662 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor roles.rbac.authorization.k8s.io

I1031 10:50:37.109779 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor networkpolicies.extensions

I1031 10:50:37.109830 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor replicasets.apps

I1031 10:50:37.109860 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor horizontalpodautoscalers.autoscaling

I1031 10:50:37.109940 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor serviceaccounts

I1031 10:50:37.109991 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor limitranges

I1031 10:50:37.110031 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor ingresses.extensions

I1031 10:50:37.110071 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor daemonsets.apps

I1031 10:50:37.110112 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor statefulsets.apps

I1031 10:50:37.110168 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor ingresses.networking.k8s.io

I1031 10:50:37.110213 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor endpoints

I1031 10:50:37.110264 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor controllerrevisions.apps

I1031 10:50:37.110309 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor events.events.k8s.io

I1031 10:50:37.110366 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor podtemplates

I1031 10:50:37.110405 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor deployments.apps

I1031 10:50:37.110447 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor jobs.batch

I1031 10:50:37.110499 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor deployments.extensions

I1031 10:50:37.110545 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor poddisruptionbudgets.policy

I1031 10:50:37.110588 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor rolebindings.rbac.authorization.k8s.io

I1031 10:50:37.110645 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor leases.coordination.k8s.io

I1031 10:50:37.110691 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor daemonsets.extensions

I1031 10:50:37.110755 1 resource\_quota\_monitor.go:228] QuotaMonitor created object count evaluatorfor networkpolicies.networking.k8s.io

I1031 10:50:37.110796 1 controllermanager.go:532] Started "resourcequota"

I1031 10:50:37.111020 1 resource\_quota\_controller.go:271] Starting resource quota controller

I1031 10:50:37.111074 1 controller\_utils.go:1029] Waiting for caches to sync for resource quota controller

I1031 10:50:37.111118 1 resource\_quota\_monitor.go:303] QuotaMonitor running

I1031 10:50:37.554884 1 controllermanager.go:532] Started "garbagecollector"

I1031 10:50:37.555326 1 garbagecollector.go:128] Starting garbage collector controller

I1031 10:50:37.555512 1 controller\_utils.go:1029] Waiting for caches to sync for garbage collectorcontroller

I1031 10:50:37.555703 1 graph\_builder.go:280] GraphBuilder running

I1031 10:50:37.578512 1 controllermanager.go:532] Started "disruption"

I1031 10:50:37.578767 1 disruption.go:333] Starting disruption controller

I1031 10:50:37.578828 1 controller\_utils.go:1029] Waiting for caches to sync for disruption controller

I1031 10:50:37.590927 1 controllermanager.go:532] Started "statefulset"

I1031 10:50:37.590985 1 stateful\_set.go:145] Starting stateful set controller

I1031 10:50:37.591176 1 controller\_utils.go:1029] Waiting for caches to sync for stateful set controller

I1031 10:50:37.634499 1 controllermanager.go:532] Started "tokencleaner"

I1031 10:50:37.634764 1 tokencleaner.go:116] Starting token cleaner controller

I1031 10:50:37.634832 1 controller\_utils.go:1029] Waiting for caches to sync for token\_cleaner controller

I1031 10:50:37.735019 1 controller\_utils.go:1036] Caches are synced for token\_cleaner controller

I1031 10:50:37.885627 1 controllermanager.go:532] Started "replicationcontroller"

I1031 10:50:37.885891 1 replica\_set.go:182] Starting replicationcontroller controller

I1031 10:50:37.885930 1 controller\_utils.go:1029] Waiting for caches to sync for ReplicationController controller

I1031 10:50:38.134490 1 controllermanager.go:532] Started "deployment"

I1031 10:50:38.134934 1 deployment\_controller.go:152] Starting deployment controller

I1031 10:50:38.135088 1 controller\_utils.go:1029] Waiting for caches to sync for deployment controller

I1031 10:50:38.833531 1 controllermanager.go:532] Started "horizontalpodautoscaling"

I1031 10:50:38.833662 1 horizontal.go:156] Starting HPA controller

I1031 10:50:38.833741 1 controller\_utils.go:1029] Waiting for caches to sync for HPA controller

I1031 10:50:39.083857 1 controllermanager.go:532] Started "csrcleaner"

I1031 10:50:39.083949 1 cleaner.go:81] Starting CSR cleaner controller

I1031 10:50:39.334482 1 controllermanager.go:532] Started "persistentvolume-binder"

I1031 10:50:39.334547 1 pv\_controller\_base.go:282] Starting persistent volume controller

I1031 10:50:39.334653 1 controller\_utils.go:1029] Waiting for caches to sync for persistent volumecontroller

I1031 10:50:39.589484 1 controllermanager.go:532] Started "attachdetach"

I1031 10:50:39.589859 1 attach\_detach\_controller.go:335] Starting attach detach controller

I1031 10:50:39.589922 1 controller\_utils.go:1029] Waiting for caches to sync for attach detach controller

I1031 10:50:39.834372 1 controllermanager.go:532] Started "endpoint"

I1031 10:50:39.834427 1 endpoints\_controller.go:166] Starting endpoint controller

I1031 10:50:39.834447 1 controller\_utils.go:1029] Waiting for caches to sync for endpoint controller

I1031 10:50:40.086068 1 controllermanager.go:532] Started "replicaset"

I1031 10:50:40.086994 1 replica\_set.go:182] Starting replicaset controller

I1031 10:50:40.087164 1 controller\_utils.go:1029] Waiting for caches to sync for ReplicaSet controller

I1031 10:50:40.333873 1 controllermanager.go:532] Started "cronjob"

I1031 10:50:40.334013 1 cronjob\_controller.go:96] Starting CronJob Manager

I1031 10:50:40.584421 1 controllermanager.go:532] Started "ttl"

W1031 10:50:40.584631 1 controllermanager.go:524] Skipping "nodeipam"

I1031 10:50:40.584749 1 core.go:170] Will not configure cloud provider routes for allocate-node-cidrs: false, configure-cloud-routes: true.

W1031 10:50:40.584854 1 controllermanager.go:524] Skipping "route"

I1031 10:50:40.584595 1 ttl\_controller.go:116] Starting TTL controller

I1031 10:50:40.584962 1 controller\_utils.go:1029] Waiting for caches to sync for TTL controller

I1031 10:50:40.734839 1 node\_lifecycle\_controller.go:77] Sending events to api server

E1031 10:50:40.735124 1 core.go:160] failed to start cloud node lifecycle controller: no cloud provider provided

W1031 10:50:40.735281 1 controllermanager.go:524] Skipping "cloud-node-lifecycle"

I1031 10:50:40.985841 1 controllermanager.go:532] Started "bootstrapsigner"

I1031 10:50:40.986080 1 controller\_utils.go:1029] Waiting for caches to sync for bootstrap\_signer controller

E1031 10:50:41.235359 1 core.go:76] Failed to start service controller: WARNING: no cloud providerprovided, services of type LoadBalancer will fail

W1031 10:50:41.235652 1 controllermanager.go:524] Skipping "service"

I1031 10:50:41.484434 1 controllermanager.go:532] Started "clusterrole-aggregation"

I1031 10:50:41.484487 1 clusterroleaggregation\_controller.go:148] Starting ClusterRoleAggregator

I1031 10:50:41.484528 1 controller\_utils.go:1029] Waiting for caches to sync for ClusterRoleAggregator controller

I1031 10:50:41.733952 1 controllermanager.go:532] Started "pvc-protection"

I1031 10:50:41.734005 1 pvc\_protection\_controller.go:100] Starting PVC protection controller

I1031 10:50:41.734023 1 controller\_utils.go:1029] Waiting for caches to sync for PVC protection controller

I1031 10:50:41.984893 1 controllermanager.go:532] Started "job"

W1031 10:50:41.984925 1 controllermanager.go:524] Skipping "ttl-after-finished"

I1031 10:50:41.985096 1 job\_controller.go:143] Starting job controller

I1031 10:50:41.985153 1 controller\_utils.go:1029] Waiting for caches to sync for job controller

I1031 10:50:42.234681 1 controllermanager.go:532] Started "podgc"

I1031 10:50:42.234752 1 gc\_controller.go:76] Starting GC controller

I1031 10:50:42.234776 1 controller\_utils.go:1029] Waiting for caches to sync for GC controller

I1031 10:50:42.485964 1 controllermanager.go:532] Started "daemonset"

I1031 10:50:42.486109 1 daemon\_controller.go:267] Starting daemon sets controller

I1031 10:50:42.486220 1 controller\_utils.go:1029] Waiting for caches to sync for daemon sets controller

I1031 10:50:42.633999 1 controllermanager.go:532] Started "csrsigning"

I1031 10:50:42.634044 1 certificate\_controller.go:113] Starting certificate controller

I1031 10:50:42.634213 1 controller\_utils.go:1029] Waiting for caches to sync for certificate controller

E1031 10:50:42.784579 1 prometheus.go:55] failed to register depth metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784612 1 prometheus.go:68] failed to register adds metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784637 1 prometheus.go:82] failed to register latency metric certificate: duplicatemetrics collector registration attempted

E1031 10:50:42.784656 1 prometheus.go:96] failed to register workDuration metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784675 1 prometheus.go:112] failed to register unfinished metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784694 1 prometheus.go:126] failed to register unfinished metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784705 1 prometheus.go:152] failed to register depth metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784718 1 prometheus.go:164] failed to register adds metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784808 1 prometheus.go:176] failed to register latency metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.784872 1 prometheus.go:188] failed to register work\_duration metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.785003 1 prometheus.go:203] failed to register unfinished\_work\_seconds metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.785027 1 prometheus.go:216] failed to register longest\_running\_processor\_microseconds metric certificate: duplicate metrics collector

registration attempted

E1031 10:50:42.785054 1 prometheus.go:139] failed to register retries metric certificate: duplicate metrics collector registration attempted

E1031 10:50:42.785069 1 prometheus.go:228] failed to register retries metric certificate: duplicate metrics collector registration attempted

I1031 10:50:42.785143 1 controllermanager.go:532] Started "csrapproving"

I1031 10:50:42.785294 1 certificate\_controller.go:113] Starting certificate controller

I1031 10:50:42.785324 1 controller\_utils.go:1029] Waiting for caches to sync for certificate controller

I1031 10:50:43.035537 1 controllermanager.go:532] Started "pv-protection"

I1031 10:50:43.036801 1 controller\_utils.go:1029] Waiting for caches to sync for resource quota controller

I1031 10:50:43.037135 1 pv\_protection\_controller.go:82] Starting PV protection controller

I1031 10:50:43.051137 1 controller\_utils.go:1029] Waiting for caches to sync for PV protection controller

I1031 10:50:43.051066 1 controller\_utils.go:1029] Waiting for caches to sync for garbage collectorcontroller

I1031 10:50:43.086760 1 controller\_utils.go:1036] Caches are synced for ReplicationController controller

W1031 10:50:43.089260 1 actual\_state\_of\_world.go:506] Failed to update statusUpdateNeeded field inactual state of world: Failed to set statusUpdateNeeded to

needed true, because nodeName="minikube" doesnot exist

I1031 10:50:43.090096 1 controller\_utils.go:1036] Caches are synced for attach detach controller

I1031 10:50:43.091024 1 controller\_utils.go:1036] Caches are synced for certificate controller

I1031 10:50:43.091074 1 controller\_utils.go:1036] Caches are synced for ClusterRoleAggregator controller

I1031 10:50:43.091117 1 controller\_utils.go:1036] Caches are synced for ReplicaSet controller

I1031 10:50:43.091197 1 controller\_utils.go:1036] Caches are synced for job controller

I1031 10:50:43.091033 1 controller\_utils.go:1036] Caches are synced for bootstrap\_signer controller

I1031 10:50:43.112846 1 controller\_utils.go:1036] Caches are synced for service account controller

I1031 10:50:43.113115 1 controller\_utils.go:1036] Caches are synced for namespace controller

I1031 10:50:43.133876 1 controller\_utils.go:1036] Caches are synced for HPA controller

I1031 10:50:43.134132 1 controller\_utils.go:1036] Caches are synced for PVC protection controller

I1031 10:50:43.134351 1 controller\_utils.go:1036] Caches are synced for certificate controller

I1031 10:50:43.134606 1 controller\_utils.go:1036] Caches are synced for endpoint controller

I1031 10:50:43.134839 1 controller\_utils.go:1036] Caches are synced for GC controller

I1031 10:50:43.136153 1 controller\_utils.go:1036] Caches are synced for deployment controller

I1031 10:50:43.146659 1 event.go:258] Event(v1.ObjectReference{Kind:"Deployment", Namespace:"kube-system", Name:"coredns", UID:"e6396714-64fe-426a-

8f46-c3aba98fb959", APIVersion:"apps/v1", ResourceVersion:"179", FieldPath:""}): type: 'Normal' reason: 'ScalingReplicaSet' Scaled up replica set coredns-

5c98db65d4 to 2

I1031 10:50:43.152205 1 controller\_utils.go:1036] Caches are synced for PV protection controller

I1031 10:50:43.167065 1 event.go:258] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"kube-system", Name:"coredns-5c98db65d4", UID:"fe1f6842-

82c3-446d-b56e-c3cb21745ecb", APIVersion:"apps/v1", ResourceVersion:"315", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: coredns-

5c98db65d4-z4rdh

I1031 10:50:43.177533 1 event.go:258] Event(v1.ObjectReference{Kind:"ReplicaSet", Namespace:"kube-system", Name:"coredns-5c98db65d4", UID:"fe1f6842-

82c3-446d-b56e-c3cb21745ecb", APIVersion:"apps/v1", ResourceVersion:"315", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: coredns-

5c98db65d4-8wztw

I1031 10:50:43.185177 1 controller\_utils.go:1036] Caches are synced for TTL controller

I1031 10:50:43.379046 1 controller\_utils.go:1036] Caches are synced for disruption controller

I1031 10:50:43.379076 1 disruption.go:341] Sending events to api server.

I1031 10:50:43.391301 1 controller\_utils.go:1036] Caches are synced for stateful set controller

I1031 10:50:43.561712 1 controller\_utils.go:1036] Caches are synced for taint controller

I1031 10:50:43.561841 1 node\_lifecycle\_controller.go:1159] Initializing eviction metric for zone:

W1031 10:50:43.561934 1 node\_lifecycle\_controller.go:833] Missing timestamp for Node minikube. Assuming now as a timestamp.

I1031 10:50:43.561965 1 node\_lifecycle\_controller.go:1059] Controller detected that zone is now in state Normal.

I1031 10:50:43.562215 1 event.go:258] Event(v1.ObjectReference{Kind:"Node", Namespace:"", Name:"minikube", UID:"77bfb13c-5ae6-475a-b6f1-d72f8447424d",

APIVersion:"", ResourceVersion:"", FieldPath:""}): type: 'Normal' reason: 'RegisteredNode' Node minikube event: Registered Node minikube in Controller

I1031 10:50:43.562272 1 taint\_manager.go:198] Starting NoExecuteTaintManager

I1031 10:50:43.587122 1 controller\_utils.go:1036] Caches are synced for daemon sets controller

I1031 10:50:43.635600 1 event.go:258] Event(v1.ObjectReference{Kind:"DaemonSet", Namespace:"kube-system", Name:"kube-proxy", UID:"2e488873-546f-

4c6e-b0b5-5f61ef6d3c86", APIVersion:"apps/v1", ResourceVersion:"204", FieldPath:""}): type: 'Normal' reason: 'SuccessfulCreate' Created pod: kube-proxy-87d7w

I1031 10:50:43.811288 1 controller\_utils.go:1036] Caches are synced for resource quota controller

I1031 10:50:43.837136 1 controller\_utils.go:1036] Caches are synced for resource quota controller

I1031 10:50:43.855040 1 controller\_utils.go:1036] Caches are synced for garbage collector controller

I1031 10:50:43.856717 1 controller\_utils.go:1036] Caches are synced for garbage collector controller

I1031 10:50:43.856732 1 garbagecollector.go:137] Garbage collector: all resource monitors have synced. Proceeding to collect garbage

I1031 10:50:43.888938 1 controller\_utils.go:1036] Caches are synced for expand controller

I1031 10:50:43.935001 1 controller\_utils.go:1036] Caches are synced for persistent volume controller

W1031 11:08:50.126849 1 reflector.go:302] k8s.io/client-go/informers/factory.go:133: watch of \*v1beta1.Event ended with: The resourceVersion for the provided

watch is too old.

==== END logs for container kube-controller-manager of pod kube-system/kube-controller-manager-minikube ====

==== START logs for container kube-proxy of pod kube-system/kube-proxy-87d7w ====

W1031 10:50:44.965842 1 server\_others.go:249] Flag proxy-mode="" unknown, assuming iptables proxy

I1031 10:50:44.978517 1 server\_others.go:143] Using iptables Proxier.

W1031 10:50:44.978941 1 proxier.go:321] clusterCIDR not specified, unable to distinguish between internal and external traffic

I1031 10:50:44.980990 1 server.go:534] Version: v1.15.0

I1031 10:50:44.990135 1 conntrack.go:100] Set sysctl 'net/netfilter/nf\_conntrack\_max' to 131072

I1031 10:50:44.990247 1 conntrack.go:52] Setting nf\_conntrack\_max to 131072

I1031 10:50:44.990547 1 conntrack.go:83] Setting conntrack hashsize to 32768

I1031 10:50:45.005880 1 conntrack.go:100] Set sysctl 'net/netfilter/nf\_conntrack\_tcp\_timeout\_established' to 86400

I1031 10:50:45.005991 1 conntrack.go:100] Set sysctl 'net/netfilter/nf\_conntrack\_tcp\_timeout\_close\_wait' to 3600

I1031 10:50:45.006162 1 config.go:96] Starting endpoints config controller

I1031 10:50:45.006200 1 controller\_utils.go:1029] Waiting for caches to sync for endpoints config controller

I1031 10:50:45.006258 1 config.go:187] Starting service config controller

I1031 10:50:45.006279 1 controller\_utils.go:1029] Waiting for caches to sync for service config controller

I1031 10:50:45.106429 1 controller\_utils.go:1036] Caches are synced for endpoints config controller

I1031 10:50:45.106432 1 controller\_utils.go:1036] Caches are synced for service config controller

==== END logs for container kube-proxy of pod kube-system/kube-proxy-87d7w ====

==== START logs for container kube-scheduler of pod kube-system/kube-scheduler-minikube ====

I1031 10:50:26.167627 1 serving.go:319] Generated self-signed cert in-memory

W1031 10:50:26.864888 1 authentication.go:249] No authentication-kubeconfig provided in order to lookup client-ca-file in configmap/extension-apiserver-

authentication in kube-system, so client certificate authentication won't work.

W1031 10:50:26.864944 1 authentication.go:252] No authentication-kubeconfig provided in order to lookup requestheader-client-ca-file in configmap/extension-

apiserver-authentication in kube-system, so request-header client certificate authentication won't work.

W1031 10:50:26.864964 1 authorization.go:146] No authorization-kubeconfig provided, so SubjectAccessReview of authorization tokens won't work.

I1031 10:50:26.870225 1 server.go:142] Version: v1.15.0

I1031 10:50:26.870439 1 defaults.go:87] TaintNodesByCondition is enabled, PodToleratesNodeTaints predicate is mandatory

W1031 10:50:26.872942 1 authorization.go:47] Authorization is disabled

W1031 10:50:26.872955 1 authentication.go:55] Authentication is disabled

I1031 10:50:26.872968 1 deprecated\_insecure\_serving.go:51] Serving healthz insecurely on [::]:10251

I1031 10:50:26.873538 1 secure\_serving.go:116] Serving securely on 127.0.0.1:10259

E1031 10:50:32.233363 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.PersistentVolumeClaim: persistentvolumeclaims is

forbidden: User "system:kube-scheduler" cannot list resource "persistentvolumeclaims" in API group "" at the cluster scope

E1031 10:50:32.233817 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1beta1.PodDisruptionBudget: poddisruptionbudgets.policy is

forbidden: User "system:kube-scheduler" cannot list resource "poddisruptionbudgets" in API group "policy" at the cluster scope

E1031 10:50:32.234188 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.StatefulSet: statefulsets.apps is forbidden: User

"system:kube-scheduler" cannot list resource "statefulsets" in API group "apps" at the cluster scope

E1031 10:50:32.234274 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.StorageClass: storageclasses.storage.k8s.io is forbidden:

User "system:kube-scheduler" cannot list resource "storageclasses" in API group "storage.k8s.io" at the cluster scope

E1031 10:50:32.234322 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.ReplicationController: replicationcontrollers is forbidden:

User "system:kube-scheduler" cannot list resource "replicationcontrollers" in API group "" at the cluster scope

E1031 10:50:32.234446 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.PersistentVolume: persistentvolumes is forbidden: User

"system:kube-scheduler" cannot list resource"persistentvolumes" in API group "" at the cluster scope

E1031 10:50:32.234631 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.ReplicaSet: replicasets.apps is forbidden: User

"system:kube-scheduler" cannot list resource "replicasets" in API group "apps" at the cluster scope

E1031 10:50:32.235092 1 reflector.go:125] k8s.io/kubernetes/cmd/kube-scheduler/app/server.go:226: Failed to list \*v1.Pod: pods is forbidden: User

"system:kube-scheduler" cannot list resource "pods" in API group "" at the cluster scope

E1031 10:50:32.235239 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.Node: nodes is forbidden: User "system:kube-scheduler"

cannot list resource "nodes" in API group ""at the cluster scope

E1031 10:50:32.235501 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.Service: services is forbidden: User "system:kube-

scheduler" cannot list resource "services" in APIgroup "" at the cluster scope

E1031 10:50:33.237175 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.StatefulSet: statefulsets.apps is forbidden: User

"system:kube-scheduler" cannot list resource "statefulsets" in API group "apps" at the cluster scope

E1031 10:50:33.237235 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.PersistentVolumeClaim: persistentvolumeclaims is

forbidden: User "system:kube-scheduler" cannot list resource "persistentvolumeclaims" in API group "" at the cluster scope

E1031 10:50:33.239827 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1beta1.PodDisruptionBudget: poddisruptionbudgets.policy is

forbidden: User "system:kube-scheduler" cannot list resource "poddisruptionbudgets" in API group "policy" at the cluster scope

E1031 10:50:33.239828 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.ReplicationController: replicationcontrollers is forbidden:

User "system:kube-scheduler" cannot list resource "replicationcontrollers" in API group "" at the cluster scope

E1031 10:50:33.240020 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.StorageClass: storageclasses.storage.k8s.io is forbidden:

User "system:kube-scheduler" cannot list resource "storageclasses" in API group "storage.k8s.io" at the cluster scope

E1031 10:50:33.242577 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.PersistentVolume: persistentvolumes is forbidden: User

"system:kube-scheduler" cannot list resource"persistentvolumes" in API group "" at the cluster scope

E1031 10:50:33.247178 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.ReplicaSet: replicasets.apps is forbidden: User

"system:kube-scheduler" cannot list resource "replicasets" in API group "apps" at the cluster scope

E1031 10:50:33.251095 1 reflector.go:125] k8s.io/kubernetes/cmd/kube-scheduler/app/server.go:226: Failed to list \*v1.Pod: pods is forbidden: User

"system:kube-scheduler" cannot list resource "pods" in API group "" at the cluster scope

E1031 10:50:33.260732 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.Node: nodes is forbidden: User "system:kube-scheduler"

cannot list resource "nodes" in API group ""at the cluster scope

E1031 10:50:33.260737 1 reflector.go:125] k8s.io/client-go/informers/factory.go:133: Failed to list \*v1.Service: services is forbidden: User "system:kube-

scheduler" cannot list resource "services" in APIgroup "" at the cluster scope

I1031 10:50:35.075558 1 leaderelection.go:235] attempting to acquire leader lease kube-system/kube-scheduler...

I1031 10:50:35.084000 1 leaderelection.go:245] successfully acquired lease kube-system/kube-scheduler

E1031 10:50:43.209356 1 factory.go:702] pod is already present in the activeQ

E1031 10:50:43.223350 1 factory.go:702] pod is already present in the activeQ

==== END logs for container kube-scheduler of pod kube-system/kube-scheduler-minikube ====

==== START logs for container storage-provisioner of pod kube-system/storage-provisioner ====

==== END logs for container storage-provisioner of pod kube-system/storage-provisioner ====

{

"kind": "EventList",

"apiVersion": "v1",

"metadata": {

"selfLink": "/api/v1/namespaces/default/events",

"resourceVersion": "1953"

},

"items": [

{

"metadata": {

"name": "minikube.15d2b5b85ca720f4",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/minikube.15d2b5b85ca720f4",

"uid": "e200a1d5-96a9-44d4-b597-2abb12207bc8",

"resourceVersion": "225",

"creationTimestamp": "2019-10-31T10:50:35Z"

},

"involvedObject": {

"kind": "Node",

"name": "minikube",

"uid": "minikube"

},

"reason": "NodeHasSufficientMemory",

"message": "Node minikube status is now: NodeHasSufficientMemory",

"source": {

"component": "kubelet",

"host": "minikube"

},

"firstTimestamp": "2019-10-31T10:50:23Z",

"lastTimestamp": "2019-10-31T10:50:23Z",

"count": 9,

"type": "Normal",

"eventTime": null,

"reportingComponent": "",

"reportingInstance": ""

},

{

"metadata": {

"name": "minikube.15d2b5b85ca7343c",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/minikube.15d2b5b85ca7343c",

"uid": "7970f9f3-5462-4875-a52a-ae9d6f8150cc",

"resourceVersion": "213",

"creationTimestamp": "2019-10-31T10:50:36Z"

},

"involvedObject": {

"kind": "Node",

"name": "minikube",

"uid": "minikube"

},

"reason": "NodeHasNoDiskPressure",

"message": "Node minikube status is now: NodeHasNoDiskPressure",

"source": {

"component": "kubelet",

"host": "minikube"

},

"firstTimestamp": "2019-10-31T10:50:23Z",

"lastTimestamp": "2019-10-31T10:50:23Z",

"count": 7,

"type": "Normal",

"eventTime": null,

"reportingComponent": "",

"reportingInstance": ""

},

{

"metadata": {

"name": "minikube.15d2b5b85ca7408f",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/minikube.15d2b5b85ca7408f",

"uid": "c248aa61-1243-4b57-a2f6-faeb2405b3c6",

"resourceVersion": "214",

"creationTimestamp": "2019-10-31T10:50:35Z"

},

"involvedObject": {

"kind": "Node",

"name": "minikube",

"uid": "minikube"

},

"reason": "NodeHasSufficientPID",

"message": "Node minikube status is now: NodeHasSufficientPID",

"source": {

"component": "kubelet",

"host": "minikube"

},

"firstTimestamp": "2019-10-31T10:50:23Z",

"lastTimestamp": "2019-10-31T10:50:23Z",

"count": 7,

"type": "Normal",

"eventTime": null,

"reportingComponent": "",

"reportingInstance": ""

},

{

"metadata": {

"name": "minikube.15d2b5bd1209ab42",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/minikube.15d2b5bd1209ab42",

"uid": "9ef47f1c-d292-4326-b238-42d45066935d",

"resourceVersion": "341",

"creationTimestamp": "2019-10-31T10:50:43Z"

},

"involvedObject": {

"kind": "Node",

"name": "minikube",

"uid": "77bfb13c-5ae6-475a-b6f1-d72f8447424d"

},

"reason": "RegisteredNode",

"message": "Node minikube event: Registered Node minikube in Controller",

"source": {

"component": "node-controller"

},

"firstTimestamp": "2019-10-31T10:50:43Z",

"lastTimestamp": "2019-10-31T10:50:43Z",

"count": 1,

"type": "Normal",

"eventTime": null,

"reportingComponent": "",

"reportingInstance": ""

},

{

"metadata": {

"name": "minikube.15d2b5bd682081cd",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/events/minikube.15d2b5bd682081cd",

"uid": "9533ffcb-56e7-4adc-8d56-b44c52bf317a",

"resourceVersion": "366",

"creationTimestamp": "2019-10-31T10:50:45Z"

},

"involvedObject": {

"kind": "Node",

"name": "minikube",

"uid": "minikube"

},

"reason": "Starting",

"message": "Starting kube-proxy.",

"source": {

"component": "kube-proxy",

"host": "minikube"

},

"firstTimestamp": "2019-10-31T10:50:45Z",

"lastTimestamp": "2019-10-31T10:50:45Z",

"count": 1,

"type": "Normal",

"eventTime": null,

"reportingComponent": "",

"reportingInstance": ""

}

]

}

{

"kind": "ReplicationControllerList",

"apiVersion": "v1",

"metadata": {

"selfLink": "/api/v1/namespaces/default/replicationcontrollers",

"resourceVersion": "1953"

},

"items": []

}

{

"kind": "ServiceList",

"apiVersion": "v1",

"metadata": {

"selfLink": "/api/v1/namespaces/default/services",

"resourceVersion": "1953"

},

"items": [

{

"metadata": {

"name": "kubernetes",

"namespace": "default",

"selfLink": "/api/v1/namespaces/default/services/kubernetes",

"uid": "cfeaf1fb-eab9-4317-a083-579bc662768c",

"resourceVersion": "149",

"creationTimestamp": "2019-10-31T10:50:35Z",

"labels": {

"component": "apiserver",

"provider": "kubernetes"

}

},

"spec": {

"ports": [

{

"name": "https",

"protocol": "TCP",

"port": 443,

"targetPort": 8443

}

],

"clusterIP": "10.96.0.1",

"type": "ClusterIP",

"sessionAffinity": "None"

},

"status": {

"loadBalancer": {}

}

}

]

}

{

"kind": "DaemonSetList",

"apiVersion": "apps/v1",

"metadata": {

"selfLink": "/apis/apps/v1/namespaces/default/daemonsets",

"resourceVersion": "1953"

},

"items": []

}

{

"kind": "DeploymentList",

"apiVersion": "apps/v1",

"metadata": {

"selfLink": "/apis/apps/v1/namespaces/default/deployments",

"resourceVersion": "1953"

},

"items": []

}

{

"kind": "ReplicaSetList",

"apiVersion": "apps/v1",

"metadata": {

"selfLink": "/apis/apps/v1/namespaces/default/replicasets",

"resourceVersion": "1953"

},

"items": []

}

{

"kind": "PodList",

"apiVersion": "v1",

"metadata": {

"selfLink": "/api/v1/namespaces/default/pods",

"resourceVersion": "1953"

},

"items": []

}

Cluster info dumped to standard output

$

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

minik

$ kubectl run

Error: required flag(s) "image" not set

Examples:

# Start a single instance of nginx.

kubectl run nginx --image=nginx

# Start a single instance of hazelcast and let the container expose port 5701 .

kubectl run hazelcast --image=hazelcast --port=5701

# Start a single instance of hazelcast and set environment variables "DNS\_DOMAIN=cluster" and "POD\_NAMESPACE=default" in the container.

kubectl run hazelcast --image=hazelcast --env="DNS\_DOMAIN=cluster" --env="POD\_NAMESPACE=default"

# Start a single instance of hazelcast and set labels "app=hazelcast" and "env=prod" in the container.

kubectl run hazelcast --image=hazelcast --labels="app=hazelcast,env=prod"

# Start a replicated instance of nginx.

kubectl run nginx --image=nginx --replicas=5

# Dry run. Print the corresponding API objects without creating them.

kubectl run nginx --image=nginx --dry-run

# Start a single instance of nginx, but overload the spec of the deployment with a partial set of values parsed from JSON.

kubectl run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": { ... } }'

# Start a pod of busybox and keep it in the foreground, don't restart it if it exits.

kubectl run -i -t busybox --image=busybox --restart=Never

# Start the nginx container using the default command, but use custom arguments (arg1 .. argN) for that command.

kubectl run nginx --image=nginx -- <arg1> <arg2> ... <argN>

# Start the nginx container using a different command and custom arguments.

kubectl run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>

# Start the perl container to compute π to 2000 places and print it out.

kubectl run pi --image=perl --restart=OnFailure -- perl -Mbignum=bpi -wle 'print bpi(2000)'

# Start the cron job to compute π to 2000 places and print it out every 5 minutes.

kubectl run pi --schedule="0/5 \* \* \* ?" --image=perl --restart=OnFailure -- perl -Mbignum=bpi -wle 'print bpi(2000)'

Options:

--allow-missing-template-keys=true: If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and

jsonpath output formats.

--attach=false: If true, wait for the Pod to start running, and then attach to the Pod as if 'kubectl attach ...' were called. Default false, unless '-i/--stdin' is set, in

which case the default is true. With '--restart=Never' the exit code of the container process is returned.

--cascade=true: If true, cascade the deletion of the resources managed by this resource (e.g. Podscreated by a ReplicationController). Default true.

--command=false: If true and extra arguments are present, use them as the 'command' field in the container, rather than the 'args' field which is the default.

--dry-run=false: If true, only print the object that would be sent, without sending it.

--env=[]: Environment variables to set in the container

--expose=false: If true, a public, external service is created for the container(s) which are run

-f, --filename=[]: to use to replace the resource.

--force=false: Only used when grace-period=0. If true, immediately remove resources from API and bypass graceful deletion. Note that immediate deletion of

some resources may result in inconsistency or data loss and requires confirmation.

--generator='': The name of the API generator to use, see http://kubernetes.io/docs/user-guide/kubectl-conventions/#generators for a list.

--grace-period=-1: Period of time in seconds given to the resource to terminate gracefully. Ignored if negative. Set to 1 for immediate shutdown. Can only be set

to 0 when --force is true (force deletion).

--hostport=-1: The host port mapping for the container port. To demonstrate a single-machine container.

--image='': The image for the container to run.

--image-pull-policy='': The image pull policy for the container. If left empty, this value will not be specified by the client and defaulted by the server

-k, --kustomize='': Process a kustomization directory. This flag can't be used together with -f or -R.

-l, --labels='': Comma separated labels to apply to the pod(s). Will override previous values.

--leave-stdin-open=false: If the pod is started in interactive mode or with stdin, leave stdin open after the first attach completes. By default, stdin will be closed

after the first attach completes.

--limits='': The resource requirement limits for this container. For example, 'cpu=200m,memory=512Mi'. Note that server side components may assign limits

depending on the server configuration, such as limit ranges.

-o, --output='': Output format. One of: json|yaml|name|go-template|go-template-file|template|templatefile|jsonpath|jsonpath-file.

--overrides='': An inline JSON override for the generated object. If this is non-empty, it is usedto override the generated object. Requires that the object supply a

valid apiVersion field.

--pod-running-timeout=1m0s: The length of time (like 5s, 2m, or 3h, higher than zero) to wait until at least one pod is running

--port='': The port that this container exposes. If --expose is true, this is also the port used by the service that is created.

--quiet=false: If true, suppress prompt messages.

--record=false: Record current kubectl command in the resource annotation. If set to false, do notrecord the command. If set to true, record the command. If not

set, default to updating the existing annotation value only if one already exists.

-R, --recursive=false: Process the directory used in -f, --filename recursively. Useful when you want to manage related manifests organized within the same

directory.

-r, --replicas=1: Number of replicas to create for this container. Default is 1.

--requests='': The resource requirement requests for this container. For example, 'cpu=100m,memory=256Mi'. Note that server side components may assign

requests depending on the server configuration, such as limit ranges.

--restart='Always': The restart policy for this Pod. Legal values [Always, OnFailure, Never]. Ifset to 'Always' a deployment is created, if set to 'OnFailure' a job is

created, if set to 'Never', a regular pod is created. For the latter two --replicas must be 1. Default 'Always', for CronJobs `Never`.

--rm=false: If true, delete resources created in this command for attached containers.

--save-config=false: If true, the configuration of current object will be saved in its annotation.Otherwise, the annotation will be unchanged. This flag is useful when

you want to perform kubectl apply on this object in the future.

--schedule='': A schedule in the Cron format the job should be run with.

--service-generator='service/v2': The name of the generator to use for creating a service. Only used if --expose is true

--service-overrides='': An inline JSON override for the generated service object. If this is non-empty, it is used to override the generated object. Requires that

the object supply a valid apiVersion field. Only used if --expose is true.

--serviceaccount='': Service account to set in the pod spec

-i, --stdin=false: Keep stdin open on the container(s) in the pod, even if nothing is attached.

--template='': Template string or path to template file to use when -o=go-template, -o=go-template-file. The template format is golang templates

[http://golang.org/pkg/text/template/#pkg-overview].

--timeout=0s: The length of time to wait before giving up on a delete, zero means determine a timeout from the size of the object

-t, --tty=false: Allocated a TTY for each container in the pod.

--wait=false: If true, wait for resources to be gone before returning. This waits for finalizers.

Usage:

kubectl run NAME --image=image [--env="key=value"] [--port=port] [--replicas=replicas] [--dry-run=bool] [--overrides=inline-json] [--command] -- [COMMAND]

[args...] [options]

Use "kubectl options" for a list of global command-line options (applies to all commands).

required flag(s) "image" not set

$

Step 3 - Deploy Containers

With a running Kubernetes cluster, containers can now be deployed.

Using kubectl run, it allows containers to be deployed onto the cluster - kubectl create deployment first-deployment --image=katacoda/docker-http-server

The status of the deployment can be discovered via the running Pods - kubectl get pods

Once the container is running it can be exposed via different networking options, depending on requirements. One possible solution is NodePort, that provides a

dynamic port to a container.

$ kubectl create deployment myfirst-deployment --image=katacoda/docker-http-server

deployment.apps/myfirst-deployment created

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

myfirst-deployment-6bdb547b7d-ghvcg 1/1 Running 0 13s

$ kubectl expose deployment myfirst-deployment --port=80 --type=NodePort

service/myfirst-deployment exposed

export PORT=$(kubectl get svc myfirst-deployment -o go-template='{{range.spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')

echo "Accessing host01:$PORT"

curl host01:$PORT

$ echo "Accessing host01:$PORT"

Accessing host01:30647

$ curl host01:$PORT

<h1>This request was processed by host: myfirst-deployment-6bdb547b7d-ghvcg</h1>

$

Step 4 - Dashboard

Enable the dashboard using Minikube with the command minikube addons enable dashboard

Make the Kubernetes Dashboard available by deploying the following YAML definition. This should only be used on Katacoda.

kubectl apply -f /opt/kubernetes-dashboard.yaml

$ kubectl apply -f /opt/kubernetes-dashboard.yaml

service/kubernetes-dashboard-katacoda created

$ ls -lrt

total 0

$ cat /opt/kubernetes-dashboard.yaml

apiVersion: v1

kind: Service

metadata:

labels:

app: kubernetes-dashboard

name: kubernetes-dashboard-katacoda

namespace: kube-system

spec:

ports:

- port: 80

protocol: TCP

targetPort: 9090

nodePort: 30000

selector:

app: kubernetes-dashboard

type: NodePort

$

The Kubernetes dashboard allows you to view your applications in a UI. In this deployment, the dashboard has been made available on port 30000 but may take a

while to start.

To see the progress of the Dashboard starting, watch the Pods within the kube-system namespace using kubectl get pods -n kube-system -w

$ kubectl get pods -n kube-system -w

NAME READY STATUS RESTARTS AGE

coredns-5c98db65d4-8wztw 1/1 Running 0 36m

coredns-5c98db65d4-z4rdh 1/1 Running 0 36m

etcd-minikube 1/1 Running 0 35m

kube-addon-manager-minikube 1/1 Running 0 35m

kube-apiserver-minikube 1/1 Running 0 35m

kube-controller-manager-minikube 1/1 Running 0 35m

kube-proxy-87d7w 1/1 Running 0 36m

kube-scheduler-minikube 1/1 Running 0 35m

kubernetes-dashboard-7b8ddcb5d6-x79sk 1/1 Running 0 2m43s

storage-provisioner 1/1 Running 0 36m

Once running, the URL to the dashboard is

master $ kubeadm init --token=102952.1a7dd4cc8d1f4cc5 --kubernetes-version $(kubeadm version -o short)

[init] Using Kubernetes version: v1.14.0

[preflight] Running pre-flight checks

[preflight] Pulling images required for setting up a Kubernetes cluster

[preflight] This might take a minute or two, depending on the speed of your internet connection

[preflight] You can also perform this action in beforehand using 'kubeadm config images pull'

[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[kubelet-start] Activating the kubelet service

[certs] Using certificateDir folder "/etc/kubernetes/pki"

[certs] Generating "ca" certificate and key

[certs] Generating "apiserver-kubelet-client" certificate and key

[certs] Generating "apiserver" certificate and key

[certs] apiserver serving cert is signed for DNS names [master kubernetes kubernetes.default kubernetes.default.svc kubernetes.default.svc.cluster.local] and IPs [10.96.0.1 172.17.0.38]

[certs] Generating "front-proxy-ca" certificate and key

[certs] Generating "front-proxy-client" certificate and key

[certs] Generating "etcd/ca" certificate and key

[certs] Generating "etcd/healthcheck-client" certificate and key

[certs] Generating "apiserver-etcd-client" certificate and key

[certs] Generating "etcd/server" certificate and key

[certs] etcd/server serving cert is signed for DNS names [master localhost] and IPs [172.17.0.38 127.0.0.1 ::1]

[certs] Generating "etcd/peer" certificate and key

[certs] etcd/peer serving cert is signed for DNS names [master localhost] and IPs [172.17.0.38 127.0.0.1::1]

[certs] Generating "sa" key and public key

[kubeconfig] Using kubeconfig folder "/etc/kubernetes"

[kubeconfig] Writing "admin.conf" kubeconfig file

[kubeconfig] Writing "kubelet.conf" kubeconfig file

[kubeconfig] Writing "controller-manager.conf" kubeconfig file

[kubeconfig] Writing "scheduler.conf" kubeconfig file

[control-plane] Using manifest folder "/etc/kubernetes/manifests"

[control-plane] Creating static Pod manifest for "kube-apiserver"

[control-plane] Creating static Pod manifest for "kube-controller-manager"

[control-plane] Creating static Pod manifest for "kube-scheduler"

[etcd] Creating static Pod manifest for local etcd in "/etc/kubernetes/manifests"

[wait-control-plane] Waiting for the kubelet to boot up the control plane as static Pods from directory "/etc/kubernetes/manifests". This can take up to 4m0s

[apiclient] All control plane components are healthy after 25.502927 seconds

[upload-config] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace

[kubelet] Creating a ConfigMap "kubelet-config-1.14" in namespace kube-system with the configuration forthe kubelets in the cluster

[upload-certs] Skipping phase. Please see --experimental-upload-certs

[mark-control-plane] Marking the node master as control-plane by adding the label "node-role.kubernetes.io/master=''"

[mark-control-plane] Marking the node master as control-plane by adding the taints [node-role.kubernetes.io/master:NoSchedule]

[bootstrap-token] Using token: 102952.1a7dd4cc8d1f4cc5

[bootstrap-token] Configuring bootstrap tokens, cluster-info ConfigMap, RBAC Roles

[bootstrap-token] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials

[bootstrap-token] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token

[bootstrap-token] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

[bootstrap-token] creating the "cluster-info" ConfigMap in the "kube-public" namespace

[addons] Applied essential addon: CoreDNS

[addons] Applied essential addon: kube-proxy

Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

You should now deploy a pod network to the cluster.

Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:

https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 172.17.0.38:6443 --token 102952.1a7dd4cc8d1f4cc5 \

--discovery-token-ca-cert-hash sha256:5223319eca3b5006a9b4d405eed3e5d9694035a5198d19f4e63802b16d63874e

master $ sudo cp /etc/kubernetes/admin.conf $HOME/

master $ sudo chown $(id -u):$(id -g) $HOME/admin.conf

master $ export KUBECONFIG=$HOME/admin.conf

master $ /

On the second node, run the command to join the cluster providing the IP address of the Master node.

kubeadm join --discovery-token-unsafe-skip-ca-verification --token=102952.1a7dd4cc8d1f4cc5 172.17.0.18:6443

This is the same command provided after the Master has been initialised.

The --discovery-token-unsafe-skip-ca-verification tag is used to bypass the Discovery Token verification. As this token is generated dynamically, we couldn't include it within the steps. When in production, use the token provided by kubeadm init.

**Step 4 - View Nodes**

The cluster has now been initialised. The Master node will manage the cluster, while our one worker node will run our container workloads.

**Task**

The Kubernetes CLI, known as *kubectl*, can now use the configuration to access the cluster. For example, the command below will return the two nodes in our cluster.

kubectl get nodes

master $ kubectl get nodes

NAME STATUS ROLES AGE VERSION

master Ready master 4m45s v1.14.0

node01 Ready <none> 45s v1.14.0

#### Step 5 - Deploy Pod

The state of the two nodes in the cluster should now be Ready. This means that our deployments can be scheduled and launched.

Using Kubectl, it's possible to deploy pods. Commands are always issued for the Master with each node only responsible for executing the workloads.

The command below create a Pod based on the Docker Image katacoda/docker-http-server.

kubectl create deployment http --image=katacoda/docker-http-server:latest

The status of the Pod creation can be viewed using kubectl get pods

Once running, you can see the Docker Container running on the node.

node01 $ docker ps | grep docker-http-server

be0b912fc705 katacoda/docker-http-server "/app" 4 seconds ago Up 4 seconds k8s\_docker-http-server\_http-7f8cbdf584-22mf7\_default\_390c2bc7-fc3d-11e9-9b25-0242ac110012\_0

node01 $

#### Step 6 - Deploy Dashboard

Kubernetes has a web-based dashboard UI giving visibility into the Kubernetes cluster.

## Task

Deploy the dashboard yaml with the command kubectl apply -f dashboard.yaml

# ------------------- Dashboard Service ------------------- #

kind: Service

apiVersion: v1

metadata:

labels:

k8s-app: kubernetes-dashboard

name: kubernetes-dashboard

namespace: kube-system

spec:

externalIPs:

- 172.17.0.31

ports:

- port: 8443

targetPort: 8443

selector:

k8s-app: kubernetes-dashboardmaster $

master $

The dashboard is deployed into the *kube-system* namespace. View the status of the deployment with kubectl get pods -n kube-system

A ServiceAccount is required to login. A ClusterRoleBinding is used to assign the new ServiceAccount (*admin-user*) the role of *cluster-admin* on the cluster.

cat <<EOF | kubectl create -f -

apiVersion: v1

kind: ServiceAccount

metadata:

name: admin-user

namespace: kube-system

---

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: ClusterRoleBinding metadata: name: admin-user

roleRef: apiGroup: rbac.authorization.k8s.io kind: ClusterRole name: cluster-admin

subjects: - kind: ServiceAccount name: admin-user namespace: kube-system

This means they can control all aspects of Kubernetes. With ClusterRoleBinding and RBAC, different level of permissions can be defined based on security requirements. More information on creating a user for the Dashboard can be found in the [Dashboard documentation](https://github.com/kubernetes/dashboard/wiki/Creating-sample-user).

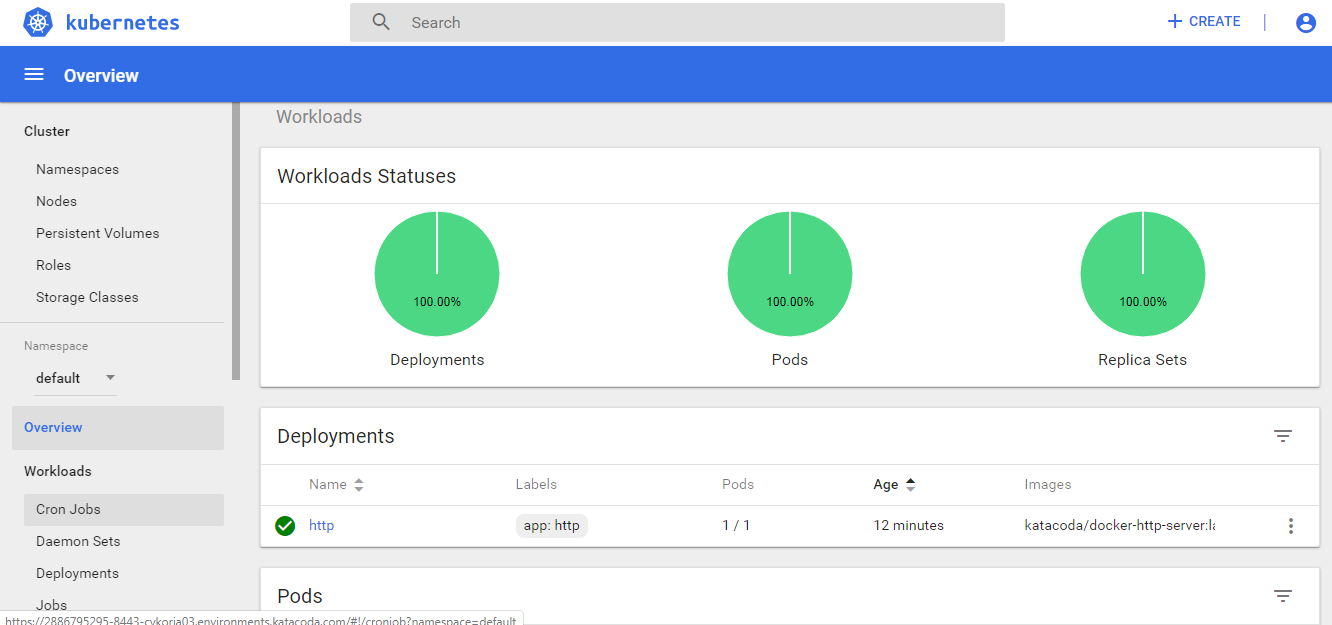
Once the ServiceAccount has been created, the token to login can be found with:

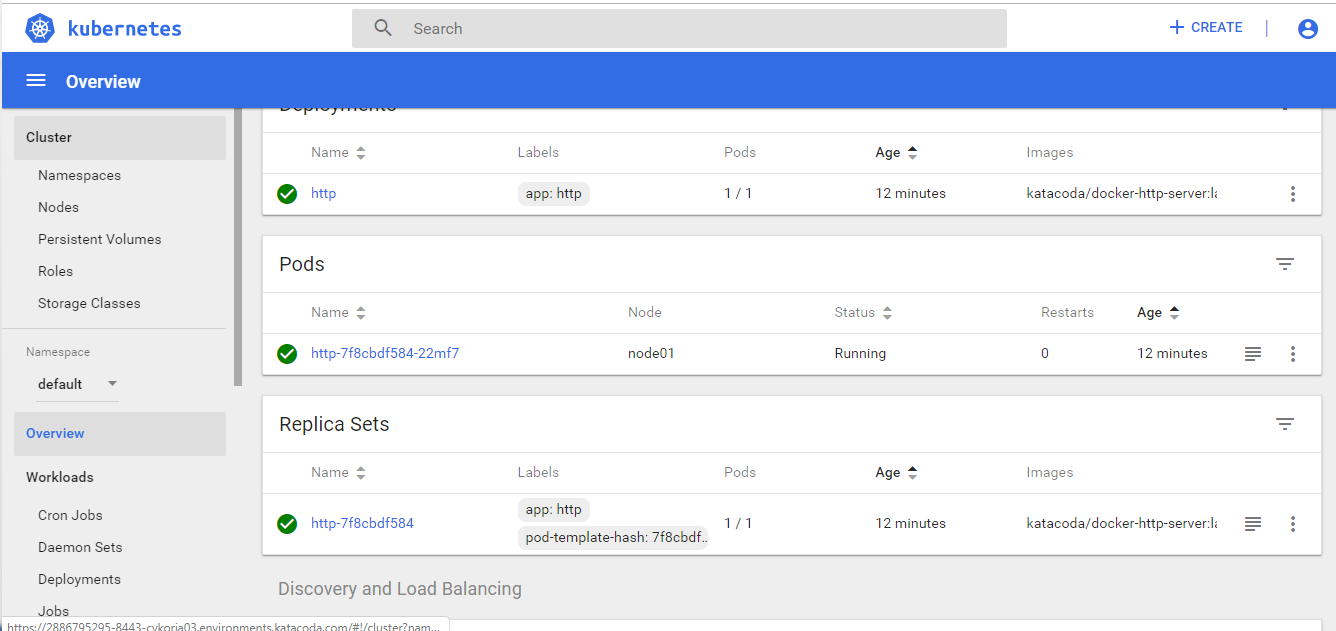
kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep admin-user | awk '{print $1}')

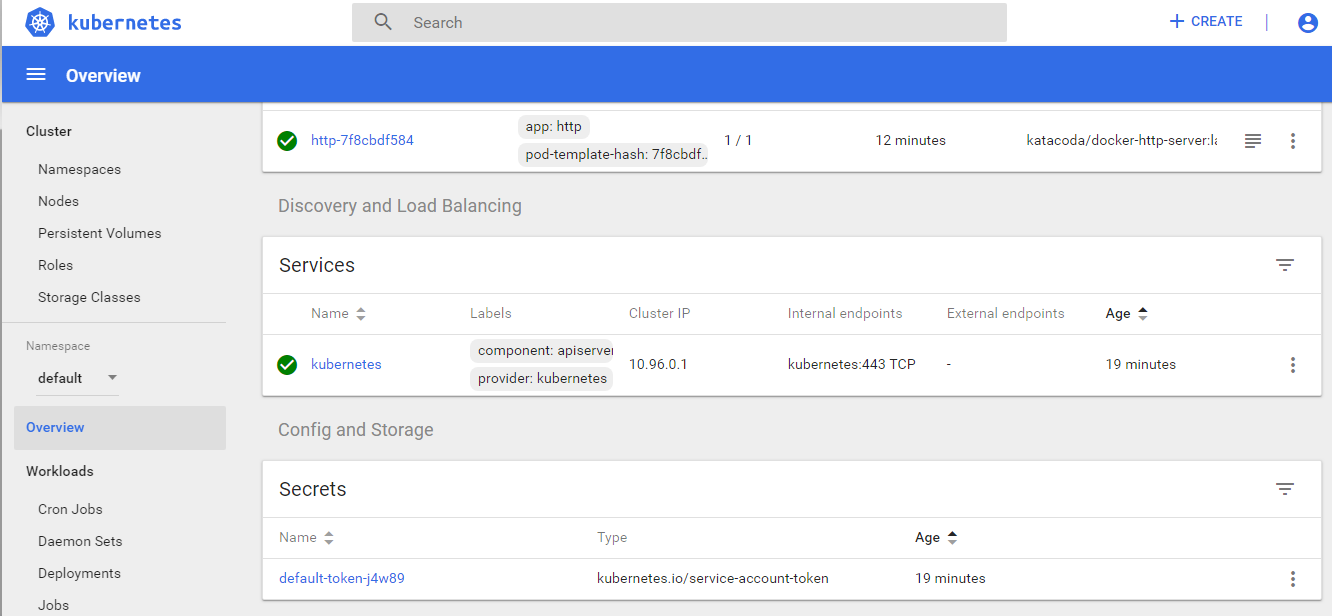
When the dashboard was deployed, it used externalIPs to bind the service to port 8443. This makes the dashboard available to outside of the cluster and viewable at <https://2886795295-8443-cykoria03.environments.katacoda.com/>

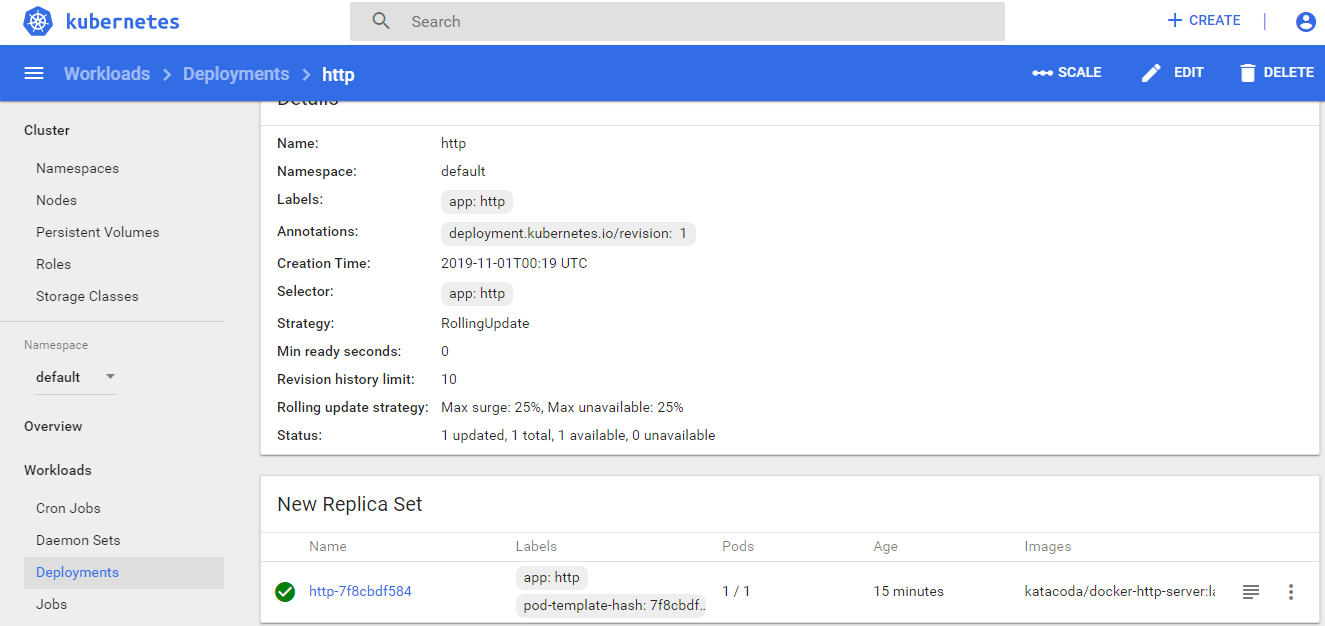
Use the admin-user token to access the dashboard.

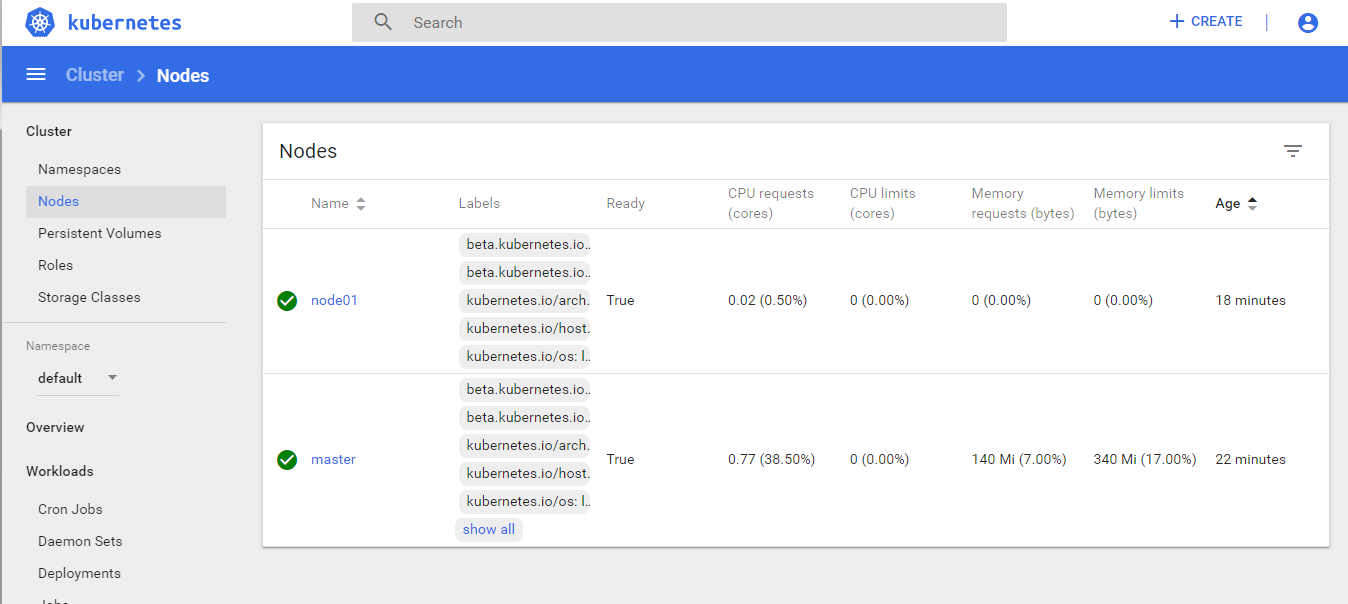
For production, instead of externalIPs, it's recommended to use kubectl proxy to access the dashboard

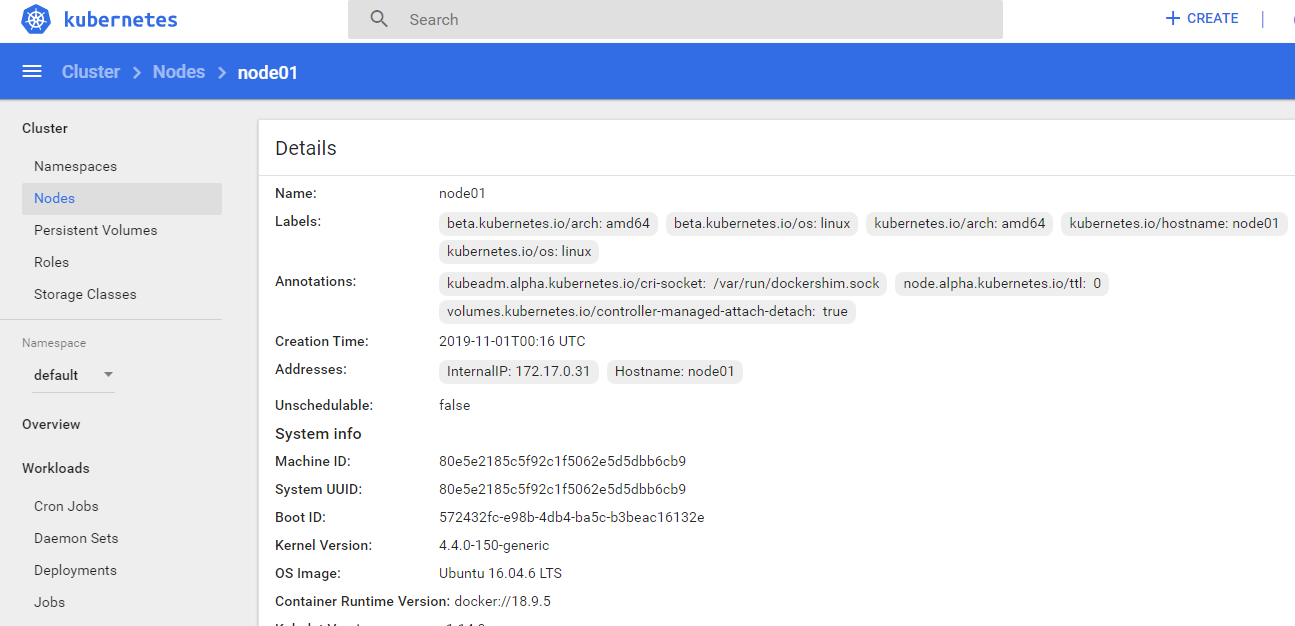


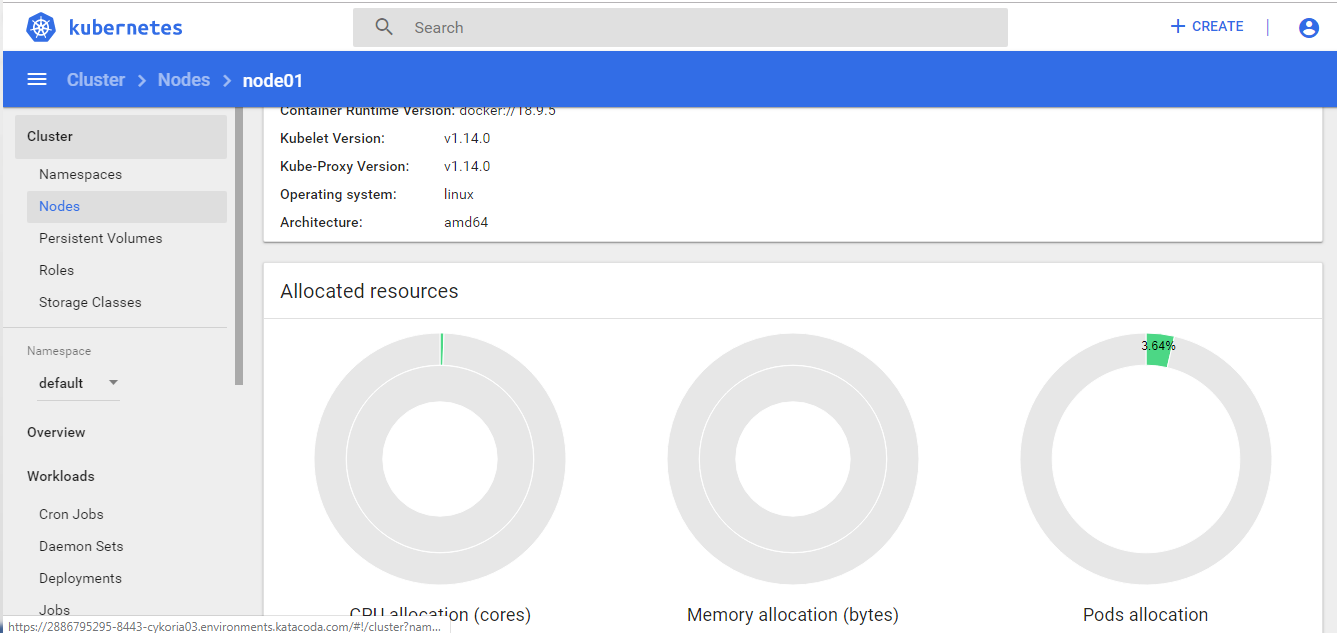


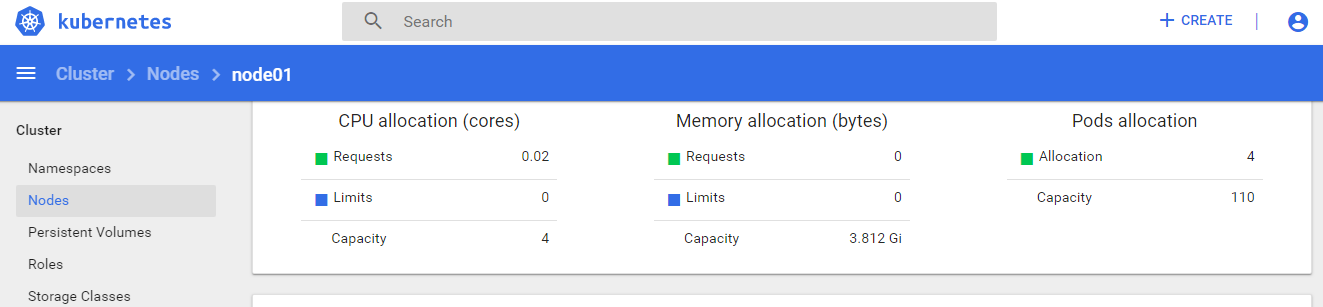


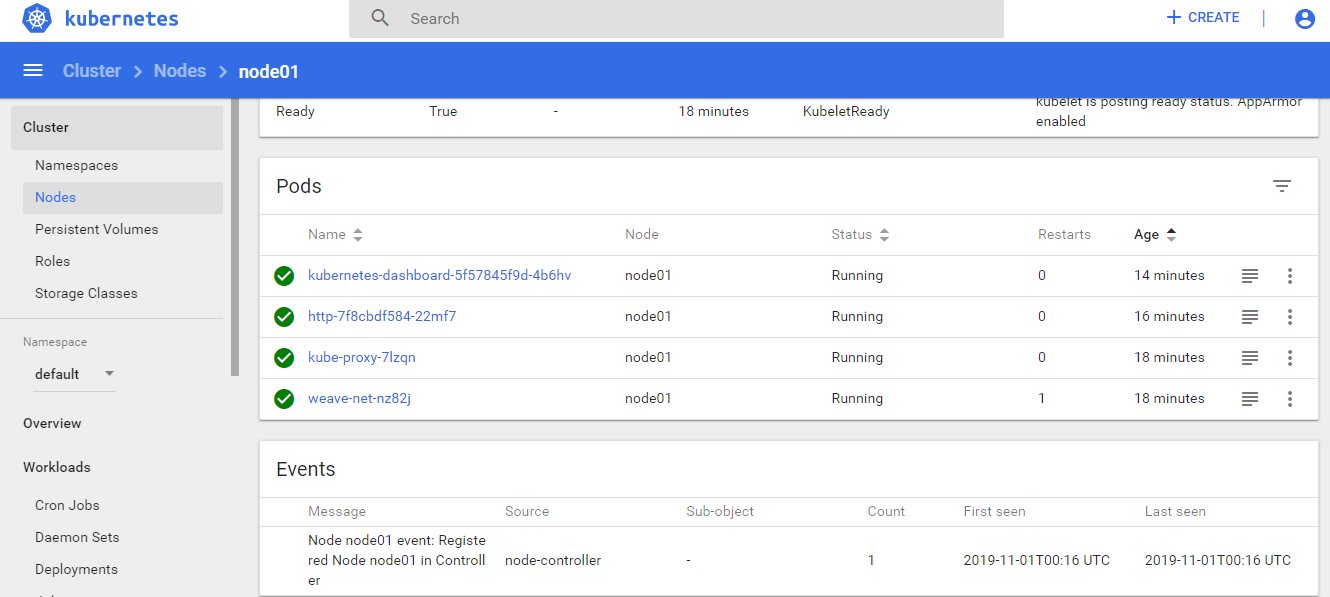












KUBECTL

our Interactive Bash Terminal. A safe place to learn and execute commands.

$ minikube start --wait=false

\* minikube v1.3.0 on Ubuntu 18.04

\* Running on localhost (CPUs=2, Memory=2461MB, Disk=47990MB) ...

\* OS release is Ubuntu 18.04.2 LTS

\* Preparing Kubernetes v1.15.0 on Docker 18.09.5 ...

- kubelet.resolv-conf=/run/systemd/resolve/resolv.conf

\* Pulling images ...

\* Launching Kubernetes ...

\* Done! kubectl is now configured to use "minikube"

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

minikube Ready master 76s v1.15.0

$ kubectl run http --image=nginx --replicas=1

kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.

deployment.apps/http created

$ kubectl get deployments

NAME READY UP-TO-DATE AVAILABLE AGE

http 1/1 1 1 21s

$ kubectl describe deployment http

Name: http

Namespace: default

CreationTimestamp: Fri, 01 Nov 2019 00:44:42 +0000

Labels: run=http

Annotations: deployment.kubernetes.io/revision: 1

Selector: run=http

Replicas: 1 desired | 1 updated | 1 total | 1 available | 0 unavailable

StrategyType: RollingUpdate

MinReadySeconds: 0

RollingUpdateStrategy: 25% max unavailable, 25% max surge

Pod Template:

Labels: run=http

Containers:

http:

Image: nginx

Port: <none>

Host Port: <none>

Environment: <none>

Mounts: <none>

Volumes: <none>

Conditions:

Type Status Reason

---- ------ ------

Available True MinimumReplicasAvailable

Progressing True NewReplicaSetAvailable

OldReplicaSets: <none>

NewReplicaSet: http-577f9d8ff5 (1/1 replicas created)

Events:

Type Reason Age From Message

---- ------ ---- ---- -------

Normal ScalingReplicaSet 35s deployment-controller Scaled up replica set http-577f9d8ff5 to 1

$

**Kubectl EXPOSE**

With the deployment created, we can use kubectl to create a service which exposes the Pods on a particular port.

Expose the newly deployed http deployment via kubectl expose. The command allows you to define the different parameters of the service and how to expose the deployment.

Use the following command to expose the container port 80 on the host 8000 binding to the external-ip of the host.

**$** kubectl expose deployment http --external-ip="172.17.0.27" --port=8000 --target-port=80

service/http exposed

You will then be able to ping the host and see the result from the HTTP service.

curl http://172.17.0.27:8000

Single command to create a deployment and expose the container port

kubectl run httpexposed --image=nginx:latest --replicas=1 --port=80 --hostport=8001

You should be able to access it using curl http://172.17.0.27:8001

Under the covers, this exposes the Pod via Docker Port Mapping. As a result, you will not see the service listed using kubectl get svc

To find the details you can use docker ps | grep httpexposed

25ac709f778c nginx "nginx -g 'daemon of…" 28 seconds ago Up 27 seconds k8s\_httpexposedlatest\_httpexposedlatest-7c6d7498f9-5qc5r\_default\_c58477af-5e0d-4202-b971-c07f03c87474\_0

e1b9426015b2 k8s.gcr.io/pause:3.1 "/pause" 33 seconds ago Up 29 seconds 0.0.0.0:8002->80/tcp k8s\_POD\_httpexposedlatest-7c6d7498f9-5qc5r\_default\_c58477af-5e0d-4202-b971-c07f03c87474\_0

6513ad6ae8f8 katacoda/docker-http-server "/app" 2 minutes ago Up 2 minutes k8s\_httpexposed\_httpexposed-569df5d86-fg4lf\_default\_817d91ab-a320-478b-ae1b-72231d0cec24\_0

e88195cb445b k8s.gcr.io/pause:3.1 "/pause" 2 minutes ago Up 2 minutes 0.0.0.0:8001->80/tcp k8s\_POD\_httpexposed-569df5d86-fg4lf\_default\_817d91ab-a320-478b-ae1b-72231d0cec24\_0

The command kubectl scale allows us to adjust the number of Pods running for a particular deployment or replication controller.

kubectl scale --replicas=3 deployment http

Listing all the pods, you should see three running for the http deployment kubectl get pods

Once each Pod starts it will be added to the load balancer service. By describing the service you can view the endpoint and the associated Pods which are included.

kubectl describe svc http

Making requests to the service will request in different nodes processing the request.

$ kubectl get pods

NAME READY STATUS RESTARTS AGE

http-577f9d8ff5-k97gd 1/1 Running 0 83s

http-577f9d8ff5-lcbhv 1/1 Running 0 83s

http-577f9d8ff5-swzsh 1/1 Running 0 12m

httpexposed-569df5d86-fg4lf 1/1 Running 0 6m53s

httpexposedlatest-7c6d7498f9-5qc5r 1/1 Running 0 4m49s

master $ kubectl cluster-info

Kubernetes master is running at https://172.17.0.66:6443

KubeDNS is running at https://172.17.0.66:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

master $ kubectl get nodes

NAME STATUS ROLES AGE VERSION

master Ready master 113m v1.14.0

node01 Ready <none> 112m v1.14.0

The first stage of launching the application is to start the Redis Master. A Kubernetes service deployment has, at least, two parts. A replication controller and a service.

The replication controller defines how many instances should be running, the Docker Image to use, and a name to identify the service. Additional options can be utilized for configuration and discovery. Use the editor above to view the YAML definition.

If Redis were to go down, the replication controller would restart it on an active node.

master $ cat redis-master-controller.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: redis-master

labels:

name: redis-master

spec:

replicas: 1

selector:

name: redis-master

template:

metadata:

labels:

name: redis-master

spec:

containers:

- name: master

image: redis:3.0.7-alpine

ports:

- containerPort: 6379

master $ cat redis-master-service.yaml

apiVersion: v1

kind: Service

metadata:

name: redis-master

labels:

name: redis-master

spec:

ports:

# the port that this service should serve on

- port: 6379

targetPort: 6379

selector:

name: redis-master

master $ cat redis-slave-controller.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: redis-slave

labels:

name: redis-slave

spec:

replicas: 2

selector:

name: redis-slave

template:

metadata:

labels:

name: redis-slave

spec:

containers:

- name: worker

image: gcr.io/google\_samples/gb-redisslave:v1

env:

- name: GET\_HOSTS\_FROM

value: dns

# If your cluster config does not include a dns service, then to

# instead access an environment variable to find the master

# service's host, comment out the 'value: dns' line above, and

# uncomment the line below.

# value: env

ports:

- containerPort: 6379

master $ kubectl create -f redis-slave-controller.yaml

replicationcontroller/redis-slave created

master $ kubectl get rc

NAME DESIRED CURRENT READY AGE

redis-master 1 1 1 6m22s

redis-slave 2 2 2 2s

master $ cat redis-slave-service.yaml

apiVersion: v1

kind: Service

metadata:

name: redis-slave

labels:

name: redis-slave

spec:

ports:

# the port that this service should serve on

- port: 6379

selector:

name: redis-slave

master $ kubectl create -f redis-slave-service.yaml

service/redis-slave created

master $

master $

master $ kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 127m

redis-master ClusterIP 10.98.30.171 <none> 6379/TCP 6m20s

redis-slave ClusterIP 10.104.240.219 <none> 6379/TCP 6s

-rw-r--r-- 1 root root 204 Nov 1 01:17 redis-slave-service.yaml

-rw-r--r-- 1 root root 717 Nov 1 01:17 redis-slave-controller.yaml

-rw-r--r-- 1 root root 228 Nov 1 01:17 redis-master-service.yaml

-rw-r--r-- 1 root root 357 Nov 1 01:17 redis-master-controller.yaml

-rw-r--r-- 1 root root 397 Nov 1 01:17 frontend-service.yaml

-rw-r--r-- 1 root root 694 Nov 1 01:17 frontend-controller.yaml

master $ cat frontend-controller.yaml

apiVersion: v1

kind: ReplicationController

metadata:

name: frontend

labels:

name: frontend

spec:

replicas: 3

selector:

name: frontend

template:

metadata:

labels:

name: frontend

spec:

containers:

- name: php-redis

image: gcr.io/google\_samples/gb-frontend:v3

env:

- name: GET\_HOSTS\_FROM

value: dns

# If your cluster config does not include a dns service, then to

# instead access environment variables to find service host

# info, comment out the 'value: dns' line above, and uncomment the

# line below.

# value: env

ports:

- containerPort: 80

master $ kubectl get pods

NAME READY STATUS RESTARTS AGE

frontend-8j2kk 1/1 Running 0 4s

frontend-q7bp7 1/1 Running 0 4s

frontend-wd8vk 1/1 Running 0 4s

redis-master-xf7gg 1/1 Running 0 13m

redis-slave-5ccc2 1/1 Running 0 6m49s

redis-slave-tpgbn 1/1 Running 0 6m49s

#### PHP Code

The PHP code uses HTTP and JSON to communicate with Redis. When setting a value requests go to redis-master while read data comes from the redis-slave nodes.

To make the frontend accessible we need to start a service to configure the proxy.

#### Start Proxy

The YAML defines the service as a NodePort. NodePort allows you to set well-known ports that are shared across your entire cluster. This is like -p 80:80 in Docker.

In this case, we define our web app is running on port 80 but we'll expose the service on 30080.

kubectl create -f frontend-service.yaml

apiVersion: v1

kind: Service

metadata:

name: redis-slave

labels:

name: redis-slave

spec:

ports:

# the port that this service should serve on

- port: 6379

selector:

name: redis-slave

kubectl get services

master $ kubectl create -f frontend-service.yaml

service/frontend created

master $ kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

frontend NodePort 10.102.78.62 <none> 80:30080/TCP 1s

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 133m

redis-master ClusterIP 10.98.30.171 <none> 6379/TCP 12m

redis-slave ClusterIP 10.104.240.219 <none> 6379/TCP 6m8s

With all controllers and services defined Kubernetes will start launching them as Pods. A pod can have different states depending on what's happening. For example, if the Docker Image is still being downloaded then the Pod will have a pending state as it's not able to launch. Once ready the status will change to running.

#### View Pods Status

You can view the status using the following command:

kubectl get pods

master $ kubectl get pods

NAME READY STATUS RESTARTS AGE

frontend-8j2kk 1/1 Running 0 2m59s

frontend-q7bp7 1/1 Running 0 2m59s

frontend-wd8vk 1/1 Running 0 2m59s

redis-master-xf7gg 1/1 Running 0 16m

redis-slave-5ccc2 1/1 Running 0 9m44s

redis-slave-tpgbn 1/1 Running 0 9m44s

#### Find NodePort

If you didn't assign a well-known NodePort then Kubernetes will assign an available port randomly. You can find the assigned NodePort using kubectl.

kubectl describe service frontend | grep NodePort

#### View UI

Once the Pod is in running state you will be able to view the UI via port 30080. Use the URL to view the page [https://2886795330-30080-cykoria04.environments.katacoda.com](https://2886795330-30080-cykoria04.environments.katacoda.com/)

Under the covers the PHP service is discovering the Redis instances via DNS. You now have a working multi-tier application deployed on Kubernetes.

**NETWORKING CAPABILITY**

* Cluster IP
* Target Ports
* NodePort
* External IPs
* Load Balancer

Kubernetes Services are an abstract that defines a policy and approach on how to access a set of Pods. The set of Pods accessed via a Service is based on a Label Selector.

Clusterip.yaml

apiVersion: v1

kind: Service

metadata:

name: webapp1-clusterip-svc

labels:

app: webapp1-clusterip

spec:

ports:

- port: 80

selector:

app: webapp1-clusterip

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp1-clusterip-deployment

spec:

replicas: 2

template:

metadata:

labels:

app: webapp1-clusterip

spec:

containers:

- name: webapp1-clusterip-pod

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

---

This will deploy a web app with two replicas to showcase load balancing along with a service. The Pods can be viewed at kubectl get pods

It will also deploy a service. kubectl get svc

More details on the service configuration and active endpoints (Pods) can be viewed via kubectl describe svc/webapp1-clusterip-svc

master $ kubectl get pods

NAME READY STATUS RESTARTS AGE

webapp1-clusterip-deployment-669c7c65c4-fgtcl 1/1 Running 0 27s

webapp1-clusterip-deployment-669c7c65c4-rlv69 1/1 Running 0 27s

master $ kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 17m

webapp1-clusterip-svc ClusterIP 10.96.25.0 <none> 80/TCP 55s

master $ kubectl describe svc/webapp1-clusterip-svc

Name: webapp1-clusterip-svc

Namespace: default

Labels: app=webapp1-clusterip

Annotations: kubectl.kubernetes.io/last-applied-configuration:

{"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app":"webapp1-clusterip"},"name":"webapp1-clusterip-svc","name...

Selector: app=webapp1-clusterip

Type: ClusterIP

IP: 10.96.25.0

Port: <unset> 80/TCP

TargetPort: 80/TCP

Endpoints: 10.32.0.5:80,10.32.0.6:80

Session Affinity: None

Events: <none>

After deploying, the service can be accessed via the ClusterIP allocated.

master $ export CLUSTER\_IP=$(kubectl get services/webapp1-clusterip-svc -o go-template='{{(index .spec.clusterIP)}}')

master $ echo CLUSTER\_IP=$CLUSTER\_IP

CLUSTER\_IP=10.96.25.0

Multiple requests will showcase how the service load balancers across multiple Pods based on the common label selector.

master $ curl $CLUSTER\_IP:80

<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-rlv69</h1>

master $ curl $CLUSTER\_IP:80

<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-fgtcl</h1>

master $

Target ports allows us to separate the port the service is available on from the port the application is listening on. TargetPort is the Port which the application is configured to listen on. Port is how the application will be accessed from the outside.

Similar to previously, the service and extra pods are deployed via kubectl apply -f clusterip-target.yaml

The following commands will create the service.

cat clusterip-target.yaml

master $ cat clusterip-target.yaml

apiVersion: v1

kind: Service

metadata:

name: webapp1-clusterip-targetport-svc

labels:

app: webapp1-clusterip-targetport

spec:

ports:

- port: 8080

targetPort: 80

selector:

app: webapp1-clusterip-targetport

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp1-clusterip-targetport-deployment

spec:

replicas: 2

template:

metadata:

labels:

app: webapp1-clusterip-targetport

spec:

containers:

- name: webapp1-clusterip-targetport-pod

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

kubectl get svc

master $ kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 26m

webapp1-clusterip-svc ClusterIP 10.96.25.0 <none> 80/TCP 9m44s

webapp1-clusterip-targetport-svc ClusterIP 10.106.132.110 <none> 8080/TCP 90s

master $

kubectl describe svc/webapp1-clusterip-targetport-svc

After the service and pods have deployed, it can be accessed via the cluster IP as before, but this time on the defined port 8080.

export CLUSTER\_IP=$(kubectl get services/webapp1-clusterip-targetport-svc -o go-template='{{(index .spec.clusterIP)}}') echo CLUSTER\_IP=$CLUSTER\_IP curl $CLUSTER\_IP:8080

curl $CLUSTER\_IP:8080

The application itself is still configured to listen on port 80. Kubernetes Service manages the translation between the two.

#### NodePort

While TargetPort and ClusterIP make it available to inside the cluster, the NodePort exposes the service on each Node’s IP via the defined static port. No matter which Node within the cluster is accessed, the service will be reachable based on the port number defined.

kubectl apply -f nodeport.yaml

When viewing the service definition, notice the additional type and NodePort property defined cat nodeport.yaml

apiVersion: v1

kind: Service

metadata:

name: webapp1-nodeport-svc

labels:

app: webapp1-nodeport

spec:

type: NodePort

ports:

- port: 80

nodePort: 30080

selector:

app: webapp1-nodeport

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp1-nodeport-deployment

spec:

replicas: 2

template:

metadata:

labels:

app: webapp1-nodeport

spec:

containers:

- name: webapp1-nodeport-pod

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

---

kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 31m

webapp1-clusterip-svc ClusterIP 10.96.25.0 <none> 80/TCP 15m

webapp1-clusterip-targetport-svc ClusterIP 10.106.132.110 <none> 8080/TCP 7m10s

webapp1-nodeport-svc NodePort 10.109.103.153 <none> 80:30080/TCP 55s

master $

kubectl describe svc/webapp1-nodeport-svc

The service can now be reached via the Node's IP

master $ kubectl describe svc/webapp1-nodeport-svc

Name: webapp1-nodeport-svc

Namespace: default

Labels: app=webapp1-nodeport

Annotations: kubectl.kubernetes.io/last-applied-configuration:

{"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app":"webapp1-nodeport"},"name":"webapp1-nodeport-svc","namesp...

Selector: app=webapp1-nodeport

Type: NodePort

IP: 10.109.103.153

Port: <unset> 80/TCP

TargetPort: 80/TCP

NodePort: <unset> 30080/TCP

Endpoints: 10.32.0.10:80,10.32.0.9:80

Session Affinity: None

External Traffic Policy: Cluster

Events: <none>

master $

The service can now be reached via the Node's IP address on the NodePort defined.

curl 172.17.0.24:30080

<h1>This request was processed by host: webapp1-nodeport-deployment-677bd89b96-sjrvs</h1>

master $

#### External IPs

Another approach to making a service available outside of the cluster is via External IP addresses.

Update the definition to the current cluster's IP address with sed -i 's/HOSTIP/172.17.0.24/g' externalip.yaml

cat externalip.yaml

apiVersion: v1

kind: Service

metadata:

name: webapp1-externalip-svc

labels:

app: webapp1-externalip

spec:

ports:

- port: 80

externalIPs:

- 172.17.0.24

selector:

app: webapp1-externalip

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp1-externalip-deployment

spec:

replicas: 2

template:

metadata:

labels:

app: webapp1-externalip

spec:

containers:

- name: webapp1-externalip-pod

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

---

kubectl apply -f externalip.yaml

kubectl get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 35m

webapp1-clusterip-svc ClusterIP 10.96.25.0 <none> 80/TCP 19m

webapp1-clusterip-targetport-svc ClusterIP 10.106.132.110 <none> 8080/TCP 11m

webapp1-externalip-svc ClusterIP 10.106.193.191 172.17.0.24 80/TCP 6s

webapp1-nodeport-svc NodePort 10.109.103.153 <none> 80:30080/TCP 4m47s

kubectl describe svc/webapp1-externalip-svc

The service is now bound to the IP address and Port 80 of the master node.

<h1>This request was processed by host: webapp1-externalip-deployment-6446b488f8-pgxdw</h1>

master $ curl 172.17.0.24

**LOAD BALANCER**

kubectl apply -f loadbalancer.yaml

The service is configured via a Load Balancer as defined in cat loadbalancer.yaml

master $ cat loadbalancer.yaml

apiVersion: v1

kind: Service

metadata:

name: webapp1-loadbalancer-svc

labels:

app: webapp1-loadbalancer

spec:

type: LoadBalancer

ports:

- port: 80

selector:

app: webapp1-loadbalancer

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp1-loadbalancer-deployment

spec:

replicas: 2

template:

metadata:

labels:

app: webapp1-loadbalancer

spec:

containers:

- name: webapp1-loadbalancer-pod

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

While the IP address is being defined, the service will show Pending. When allocated, it will appear in the service list.

kubectl get svc

kubectl describe svc/webapp1-loadbalancer-svc

The service can now be accessed via the IP address assigned, in this case from the 10.10.0.0/26 range.

export LoadBalancerIP=$(kubectl get services/webapp1-loadbalancer-svc -o go-template='{{(index .status.loadBalancer.ingress 0).ip}}') echo LoadBalancerIP=$LoadBalancerIP curl $LoadBalancerIP

curl $LoadBalancerIP

**INGRESS ROUTING**

Kubernetes have advanced networking capabilities that allow Pods and Services to communicate inside the cluster's network. An Ingress enables inbound connections to the cluster, allowing external traffic to reach the correct Pod.

Ingress enables externally-reachable urls, load balance traffic, terminate SSL, offer name based virtual hosting for a Kubernetes cluster.

In this scenario you will learn how to deploy and configure Ingress rules to manage incoming HTTP requests.

#### Step 1 - Create Deployment

To start, deploy an example HTTP server that will be the target of our requests. The deployment contains three deployments, one called webapp1 and a second called webapp2, and a third called webapp3 with a service for each.

cat deployment.yaml

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp1

spec:

replicas: 1

template:

metadata:

labels:

app: webapp1

spec:

containers:

- name: webapp1

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp2

spec:

replicas: 1

template:

metadata:

labels:

app: webapp2

spec:

containers:

- name: webapp2

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: webapp3

spec:

replicas: 1

template:

metadata:

labels:

app: webapp3

spec:

containers:

- name: webapp3

image: katacoda/docker-http-server:latest

ports:

- containerPort: 80

---

apiVersion: v1

kind: Service

metadata:

name: webapp1-svc

labels:

app: webapp1

spec:

ports:

- port: 80

selector:

app: webapp1

---

apiVersion: v1

kind: Service

metadata:

name: webapp2-svc

labels:

app: webapp2

spec:

ports:

- port: 80

selector:

app: webapp2

---

apiVersion: v1

kind: Service

metadata:

name: webapp3-svc

labels:

app: webapp3

spec:

ports:

- port: 80

selector:

app: webapp3

## Task

Deploy the definitions with kubectl apply -f deployment.yaml

deployment.extensions/webapp1 created

deployment.extensions/webapp2 created

deployment.extensions/webapp3 created

service/webapp1-svc created

service/webapp2-svc created

service/webapp3-svc created

The status can be viewed with kubectl get deployment

NAME READY UP-TO-DATE AVAILABLE AGE

webapp1 1/1 1 1 46s

webapp2 1/1 1 1 46s

webapp3 1/1 1 1 46s

#### Step 2 - Deploy Ingress

The YAML file ingress.yaml defines a Nginx-based Ingress controller together with a service making it available on Port 80 to external connections using ExternalIPs. If the Kubernetes cluster was running on a cloud provider then it would use a LoadBalancer service type.

The ServiceAccount defines the account with a set of permissions on how to access the cluster to access the defined Ingress Rules. The default server secret is a self-signed certificate for other Nginx example SSL connections and is required by the [Nginx Default Example](https://github.com/nginxinc/kubernetes-ingress/tree/master/deployments).

cat ingress.yaml

apiVersion: v1

kind: Namespace

metadata:

name: nginx-ingress

---

apiVersion: v1

kind: Secret

metadata:

name: default-server-secret

namespace: nginx-ingress

type: Opaque

data:

tls.crt: 

tls.key: 

---

apiVersion: v1

kind: ServiceAccount

metadata:

name: nginx-ingress

namespace: nginx-ingress

---

kind: ConfigMap

apiVersion: v1

metadata:

name: nginx-config

namespace: nginx-ingress

data:

---

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: nginx-ingress

namespace: nginx-ingress

spec:

replicas: 1

selector:

matchLabels:

app: nginx-ingress

template:

metadata:

labels:

app: nginx-ingress

spec:

serviceAccountName: nginx-ingress

containers:

- image: nginx/nginx-ingress:edge

imagePullPolicy: Always

name: nginx-ingress

ports:

- name: http

containerPort: 80

- name: https

containerPort: 443

env:

- name: POD\_NAMESPACE

valueFrom:

fieldRef:

fieldPath: metadata.namespace

- name: POD\_NAME

valueFrom:

fieldRef:

fieldPath: metadata.name

args:

- -nginx-configmaps=$(POD\_NAMESPACE)/nginx-config

- -default-server-tls-secret=$(POD\_NAMESPACE)/default-server-secret

---

apiVersion: v1

kind: Service

metadata:

name: nginx-ingress

namespace: nginx-ingress

spec:

type: NodePort

ports:

- port: 80

targetPort: 80

protocol: TCP

name: http

- port: 443

targetPort: 443

protocol: TCP

name: https

selector:

app: nginx-ingress

externalIPs:

- 172.17.0.40

## Task

The Ingress controllers are deployed in a familiar fashion to other Kubernetes objects with kubectl create -f ingress.yaml

ingress.extensions/webapp-ingress created

master $ kubectl get ing

NAME HOSTS ADDRESS PORTS AGE

webapp-ingress my.kubernetes.example 80 4m7s

The status can be identified using kubectl get deployment -n nginx-ingress

#### Step 3 - Deploy Ingress Rules

Ingress rules are an object type with Kubernetes. The rules can be based on a request host (domain), or the path of the request, or a combination of both.

An example set of rules are defined within cat ingress-rules.yaml

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

name: webapp-ingress

spec:

rules:

- host: my.kubernetes.example

http:

paths:

- path: /webapp1

backend:

serviceName: webapp1-svc

servicePort: 80

- path: /webapp2

backend:

serviceName: webapp2-svc

servicePort: 80

- backend:

serviceName: webapp3-svc

servicePort: 80

The important parts of the rules are defined below.

The rules apply to requests for the host my.kubernetes.example. Two rules are defined based on the path request with a single catch all definition. Requests to the path /webapp1 are forwarded onto the service webapp1-svc. Likewise, the requests to /webapp2 are forwarded to webapp2-svc. If no rules apply, webapp3-svc will be used.

This demonstrates how an application's URL structure can behave independently about how the applications are deployed.

- host: my.kubernetes.example

http:

paths:

- path: /webapp1

backend:

serviceName: webapp1-svc

servicePort: 80

- path: /webapp2

backend:

serviceName: webapp2-svc

servicePort: 80

- backend:

serviceName: webapp3-svc

servicePort: 80

## Task

As with all Kubernetes objects, they can be deployed via kubectl create -f ingress-rules.yaml

Once deployed, the status of all the Ingress rules can be discovered via kubectl get ing

**Step 4 - Test**

With the Ingress rules applied, the traffic will be routed to the defined place.

The first request will be processed by the *webapp1* deployment.

curl -H "Host: my.kubernetes.example" 172.17.0.40/webapp1

The second request will be processed by the *webapp2* deployment.

curl -H "Host: my.kubernetes.example" 172.17.0.40/webapp2

Finally, all other requests will be processed by *webapp3* deployment.

curl -H "Host: my.kubernetes.example" 172.17.0.40

master $ curl -H "Host: my.kubernetes.example" 172.17.0.40/webapp1

<h1>This request was processed by host: webapp1-6d7df9f8d-hs24p</h1>

master $ curl -H "Host: my.kubernetes.example" 172.17.0.40/webapp2

<h1>This request was processed by host: webapp2-6d48b8ff76-lgnfb</h1>

master $ curl -H "Host: my.kubernetes.example" 172.17.0.40

<h1>This request was processed by host: webapp3-7df59dc67b-hgjv7</h1>

master $

### Liveness and Readiness Healthchecks

#### Step 1 - Launch Cluster

To start, we need to launch a Kubernetes cluster.

Execute the command below to start the cluster components and download the Kubectl CLI.

launch.sh

After the cluster has started, deploy the demo application with kubectl apply -f deploy.yaml

kind: List

apiVersion: v1

items:

- kind: ReplicationController

apiVersion: v1

metadata:

name: frontend

labels:

name: frontend

spec:

replicas: 1

selector:

name: frontend

template:

metadata:

labels:

name: frontend

spec:

containers:

- name: frontend

image: katacoda/docker-http-server:health

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 1

timeoutSeconds: 1

livenessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 1

timeoutSeconds: 1

- kind: ReplicationController

apiVersion: v1

metadata:

name: bad-frontend

labels:

name: bad-frontend

spec:

replicas: 1

selector:

name: bad-frontend

template:

metadata:

labels:

name: bad-frontend

spec:

containers:

- name: bad-frontend

image: katacoda/docker-http-server:unhealthy

readinessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 1

timeoutSeconds: 1

livenessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 1

timeoutSeconds: 1

- kind: Service

apiVersion: v1

metadata:

labels:

app: frontend

kubernetes.io/cluster-service: "true"

name: frontend

spec:

type: NodePort

ports:

- port: 80

nodePort: 30080

selector:

app: frontend

#### Step 2 - Readiness Probe

When deploying the cluster, two pods were also deployed to demonstrate health checking. You can view the deployment with cat deploy.yaml.

When deploying the Replication Controller, each Pod has a Readiness and Liveness check. Each check has the following format for performing a healthcheck over HTTP.

livenessProbe:

httpGet:

path: /

port: 80

initialDelaySeconds: 1

timeoutSeconds: 1

The settings can be changed to call different endpoints, for example, /ping, based on your application.

## Get Status

The first Pod, bad-frontend is an HTTP service which always returns a 500 error indicating it hasn't started correctly. You can view the status of the Pod with kubectl get pods --selector="name=bad-frontend"

Kubectl will return the Pods deployed with our particular label. Because the healthcheck is failing, it will say that zero containers are ready. It will also indicate the number of restart attempts of the container.

To find out more details of why it's failing, describe the Pod.

To find out more details of why it's failing, describe the Pod.

pod=$(kubectl get pods --selector="name=bad-frontend" --output=jsonpath={.items..metadata.name}) kubectl describe pod $pod

## Readiness OK

Our second Pod, frontend, returns an OK status on launch.

kubectl get pods --selector="name=frontend"

#### Step 3 - Liveness Probe

With our second Pod currently running in a health state, we can simulate a failure occurring.

At present, no crashes should have occurred. kubectl get pods --selector="name=frontend"

## Crash Service

The HTTP server has an additional endpoint that will cause it to return 500 errors. Using kubectl exec it's possible to call the endpoint.

pod=$(kubectl get pods --selector="name=frontend" --output=jsonpath={.items..metadata.name}) kubectl exec $pod -- /usr/bin/curl -s localhost/unhealthy

## Liveness

Based on the configuration, Kubernetes will execute the Liveness Probe. If the Probe fails, Kubernetes will destroy and re-create the failed container. Execute the above command to crash the service and watch Kubernetes recover it automatically.

kubectl get pods --selector="name=frontend"

The check may take a few moments to detect

**SECRETS MANAGEMENT**

Kubernetes requires secrets to be encoded as Base64 strings.

Using the command line tool we can create the Base64 strings and store them as variables to use in a file. username=$(echo -n "admin" | base64) password=$(echo -n "a62fjbd37942dcs" | base64)

The secret is defined using yaml. Below we'd using the variables defined above and providing them with friendly labels which our application can use. This will create a collection of key/value secrets that can be accessed via the name, in this case test-secret

echo "apiVersion: v1 kind: Secret metadata: name: test-secret type: Opaque data: username: $username password: $password" >> secret.yaml

This yaml file can be used to with Kubectl to create our secret. When launching pods that require access to the secret we'll refer to the collection via the friendly-name.

## Task: Create the secret

Use kubectl to create our secret.

kubectl create -f secret.yaml

The following command allows you to view all the secret collections defined.

kubectl get secrets

NAME TYPE DATA AGE

default-token-hpnrn kubernetes.io/service-account-token 3 5m17s

test-secret Opaque 2 11s

master $

#### Consume via Environment Variables

In the file secret-env.yaml we've defined a Pod which has environment variables populated from the previously created secret.

View the file using cat secret-env.yaml

To populate the environment variable we define the name, in this case SECRET\_USERNAME, along with the name of the secrets collection and the key which containers the data.

The structure looks like this:

- name: SECRET\_USERNAME

valueFrom:

secretKeyRef:

name: test-secret

key: username

## Task

Launch the Pod using kubectl create -f secret-env.yaml

Once the Pod started, you output the populated environment variables. kubectl exec -it secret-env-pod env | grep SECRET\_

Kubernetes decodes the base64 value when populating the environment variables. You should see the original username/password combination we defined. These variables can now be used for accessing APIs, Databases etc.

You can check the status of a Pod using kubectl get pods.

In the next step we'll mount the secrets as files.