Workloads: Deployments and Jobs

Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

03 Scale clusters

04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

06 Get software into a cluster

Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

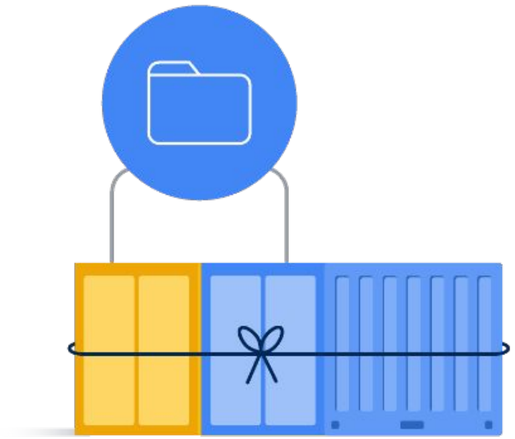
03 Scale clusters

04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

06 Get software into a cluster

Pods, Deployments, and controllers



Pods

Tightly coupled containers that share resources.

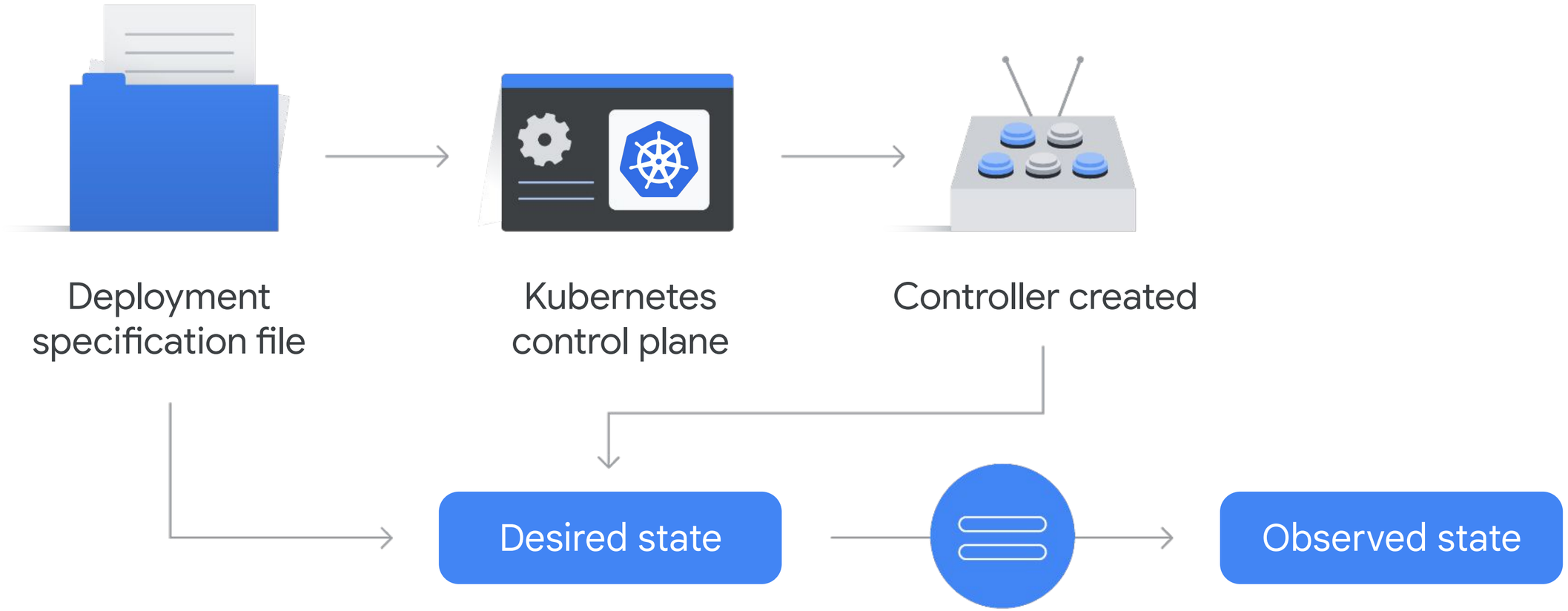
Deployment

A Kubernetes resource that describes the desired state of Pods.

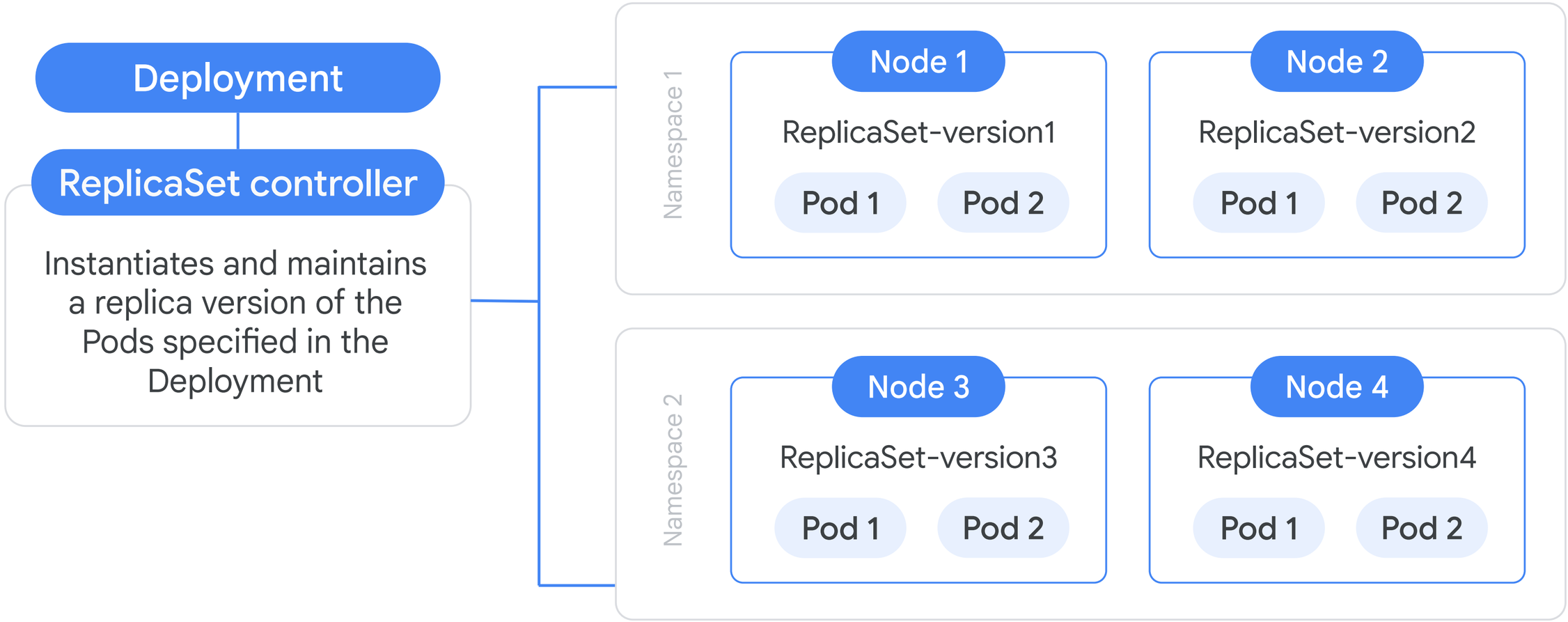
Controller

A process that ensures that the desired state of objects match the observed state.

The deployment process



The Deployment creates and configures a ReplicaSet



Deployment object file details

```
apiVersion: apps/v1
kind: Deployment
metadata:
   name: my-app
spec:
   replicas: 3
   template:
      metadata:
        labels:
          app: my-app
      spec:
         containers:
         - name: my-app
           image: gcr.io/demo/my-app:1.0
           ports:
           - containerPort: 8080
```

Written in YAML and typically identifies:

- The API version
- The kind, which in this case is a Deployment
- The name of the Deployment
- The number of Pod replicas
- A Pod template, which defines the metadata and specifications of each of the Pods in the ReplicaSet
- A container image
- A specific port to expose and accept traffic for the container

Deployment lifecycle states

Processing state

Indicates what task is being performed.

Complete state

Indicates that all new replicas are available and updated.

Failed state

Indicates that the creation of a new ReplicaSet could not be completed.

Deployments can be used for Pod management

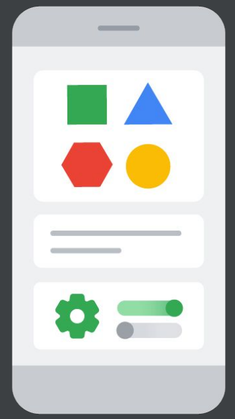
Updating

Rolling back

Scaling or Autoscaling

Deployments are designed for stateless applications

Stateless applications



Don’t store data or state in a cluster or persistent storage.

- Can be scaled up or down as needed without impacting the application's functionality.
- Examples include:
  - API servers that provide access to data.
  - Websites that don’t contain dynamic content.

Create a Deployment: Options 1 and 2

1

```
$ kubectl apply -f \
[DEPLOYMENT_FILE]
```

It's a declarative method because:

- You specify the desired state.
- The Kubernetes API server creates the objects to achieve that state.

2

```
$ kubectl create deployment \
[DEPLOYMENT_NAME] \
  --image [IMAGE]:[TAG] \
  --replicas 3 \
  --labels [KEY]=[VALUE] \
  --port 8080 \
  --generator deployment/apps.v1 \
  --save-config
```

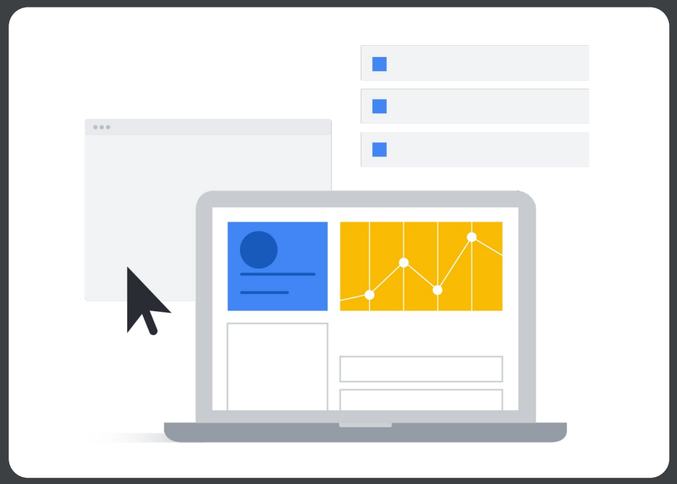
You must specify:

- The image and tag to run in the container.
- How many replicas to launch.
- Which port to expose.
- Which API to use.
- Whether to save the configuration for future use.

Create a Deployment: Option 3

3

Google Cloud console



The GKE Workloads menu > **Create a Deployment**

You can include:

- A container image
- Environment variables
- Initialization commands

Option to display the Deployment specifications in YAML format.

Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

03 Scale clusters

04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

06 Get software into a cluster

Two commands to inspect the state of a Deployment

```
$ kubectl get deployment [DEPLOYMENT NAME]
```

```
$ kubectl describe deployment [DEPLOYMENT NAME]
```

kubectl get deployment command

```
$ kubectl get deployment [DEPLOYMENT NAME]
```

```
$ kubectl describe deployment [DEPLOYMENT NAME]
```

```
master $ kubectl get deployment nginx-deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
Nginx-deployment    3/3     3            3           3m
```

- **READY**: Displays the number of replicas in the Deployment specification and how many replicas are currently running.
- **UP-TO-DATE**: Displays the number of replicas that are fully up to date.
- **AVAILABLE**: Displays the number of replicas available to the users.
- **AGE**: Displays the time the replicas have been available to the users.

kubectl get deployment command

```
$ kubectl get deployment [DEPLOYMENT NAME]
```

```
$ kubectl describe deployment [DEPLOYMENT NAME]
```

```
master $ kubectl get deployment nginx-deployment
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
Nginx-deployment    3/3     3            3           3
```

```
$ kubectl get deployment [DEPLOYMENT NAME] -o yaml > this.yaml
```

Useful for making a Deployment a permanent, managed part of your infrastructure.

kubectl describe command

```
$ kubectl get deployment [DEPLOYMENT NAME]
```

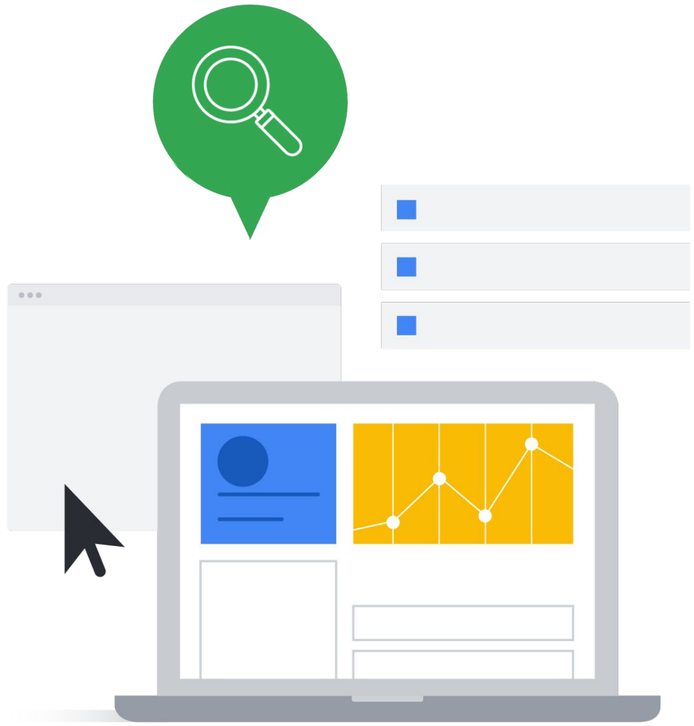
```
$ kubectl describe deployment [DEPLOYMENT NAME]
```

Includes the Deployment's:

- Current state
- Desired state
- Replicas
- Selector

```
master $ kubectl get deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Thur, 4 Apr 2024 15:23:46 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
Pod Template:
   Labels: app=nginx
   Containers
     nginx:
      Image:        nginx:1.15.4
      Port:         80/TCP
      Host Port:    0/TCP
```

Inspect through the Google Cloud console



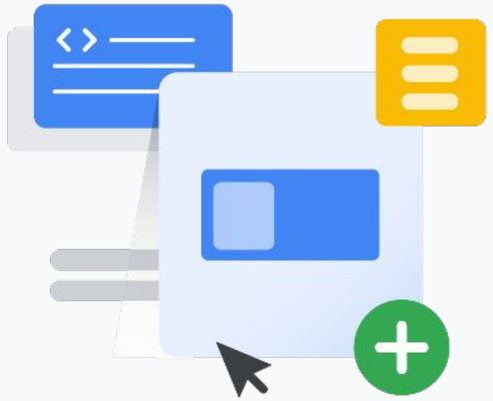
Google Cloud console

Displayed is the:

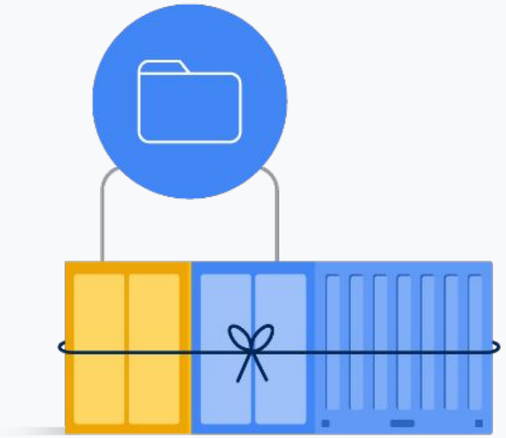
- Deployment name
- Date the Deployment was created
- Desired and current state
- Number of replicas
- Selector being used to match Pods
- List of events that have occurred

Deployment update

Example



You updated a container image with new features



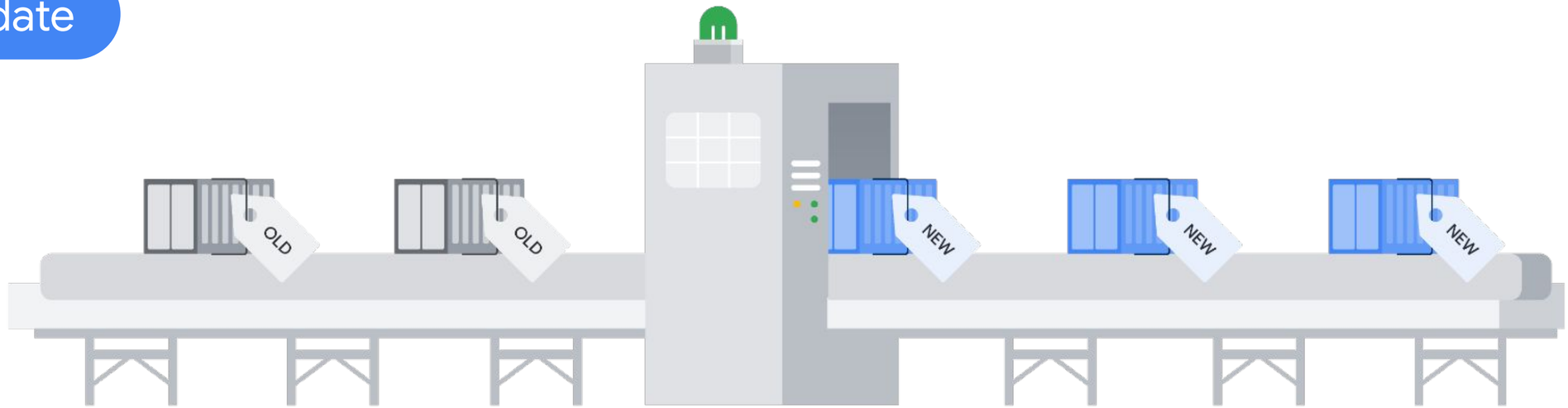
You need to update your Pod with the new version

Ways to update a Deployment

1	`$ kubectl apply -f [DEPLOYMENT_FILE]`	Update deployment specifications outside the Pod template.
2	`$ kubectl set image deployment [DEPLOYMENT_NAME] [IMAGE] [IMAGE]:[TAG]`	Make changes to the Pod template specifications.
3	`$ kubectl edit \ deployment/[DEPLOYMENT_NAME]`	Make changes directly in the specification file.
4	Google Cloud console	

Rolling updates, aka a “ramped strategy”

App update



New set of Pods are launched in a ReplicaSet

Old Pods are gracefully retired

GKE updates Pods in a Deployment one at a time to avoid downtime or disruptions

Configuring a rolling update

`maxSurge`

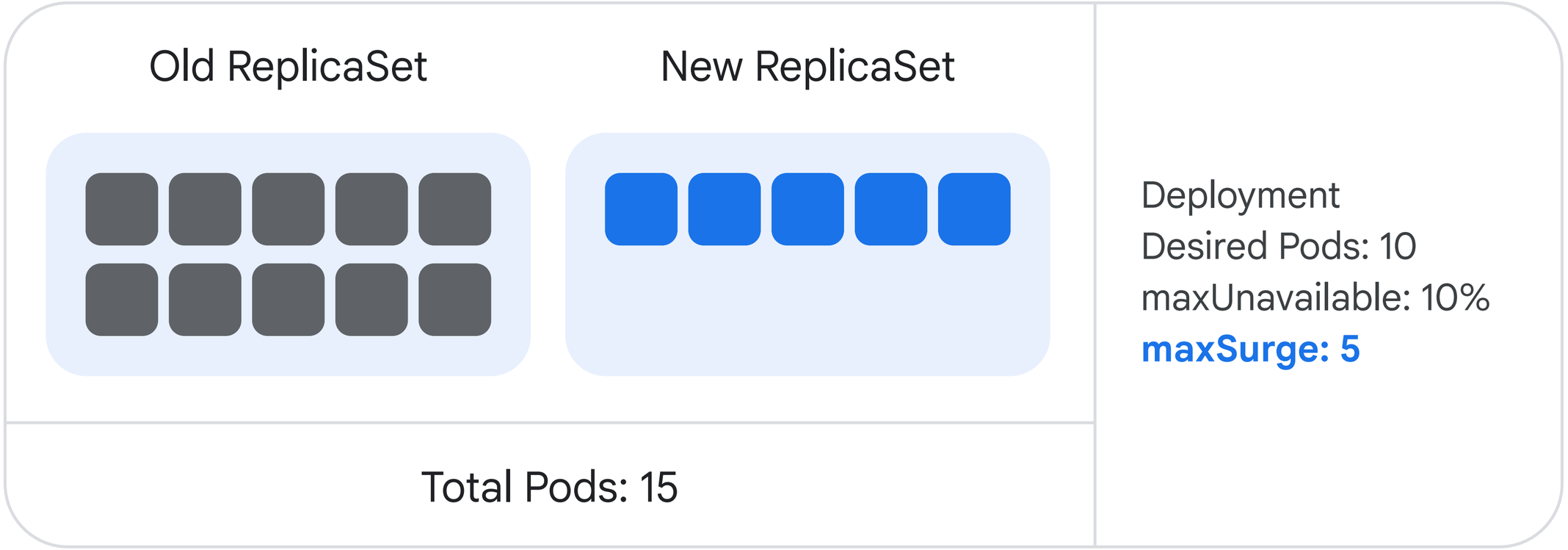
Specifies the maximum number of extra Pods that can be simultaneously running on the new version.

`maxUnavailable`

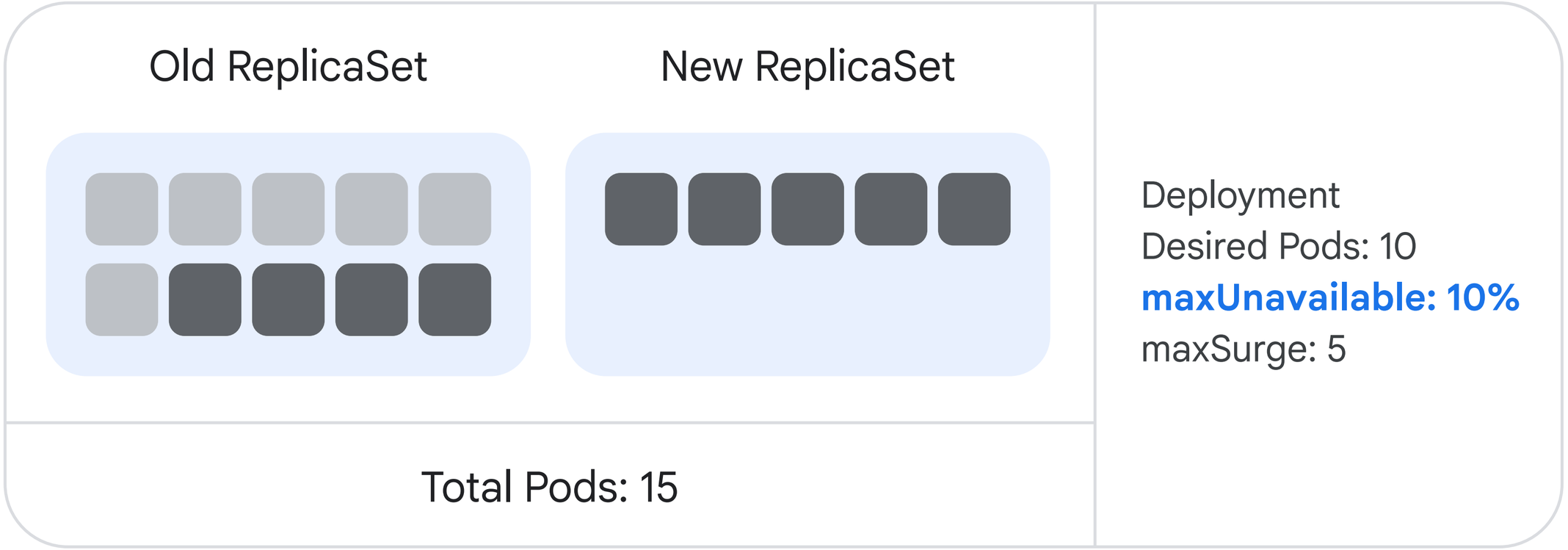
Specifies the maximum number of Pods that can be unavailable at the same time.

The maximum values can either be absolutes or percentages.

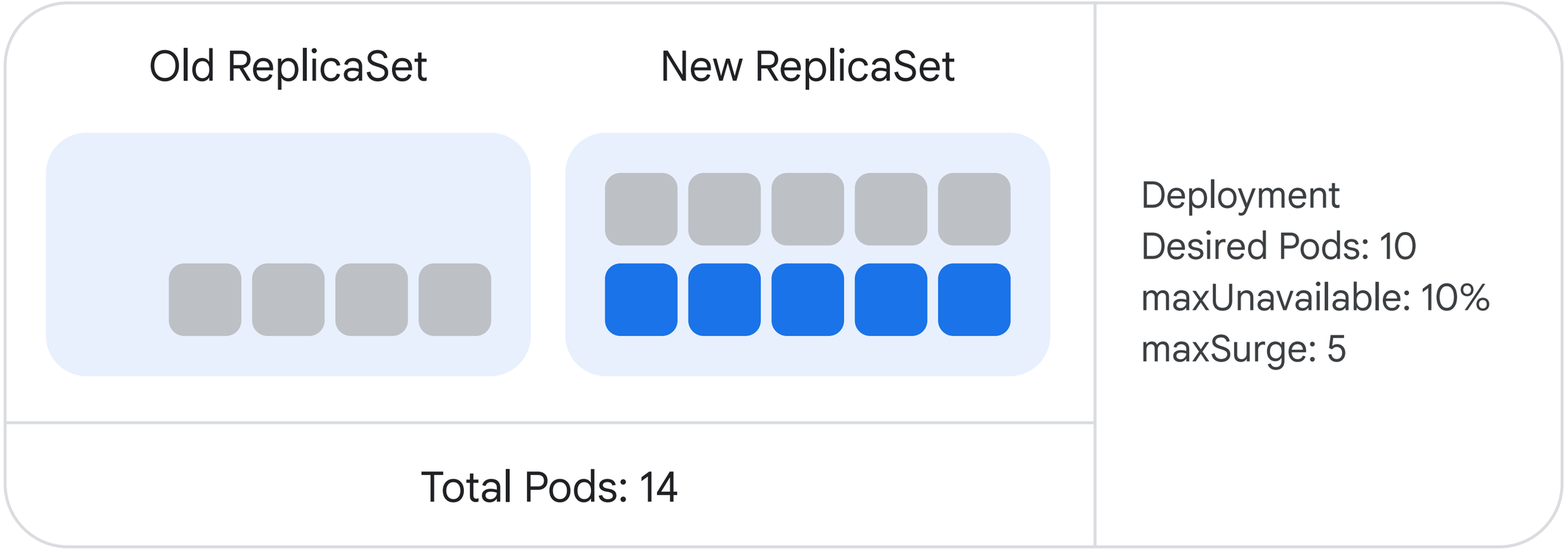
New Pods are created in a new ReplicaSet



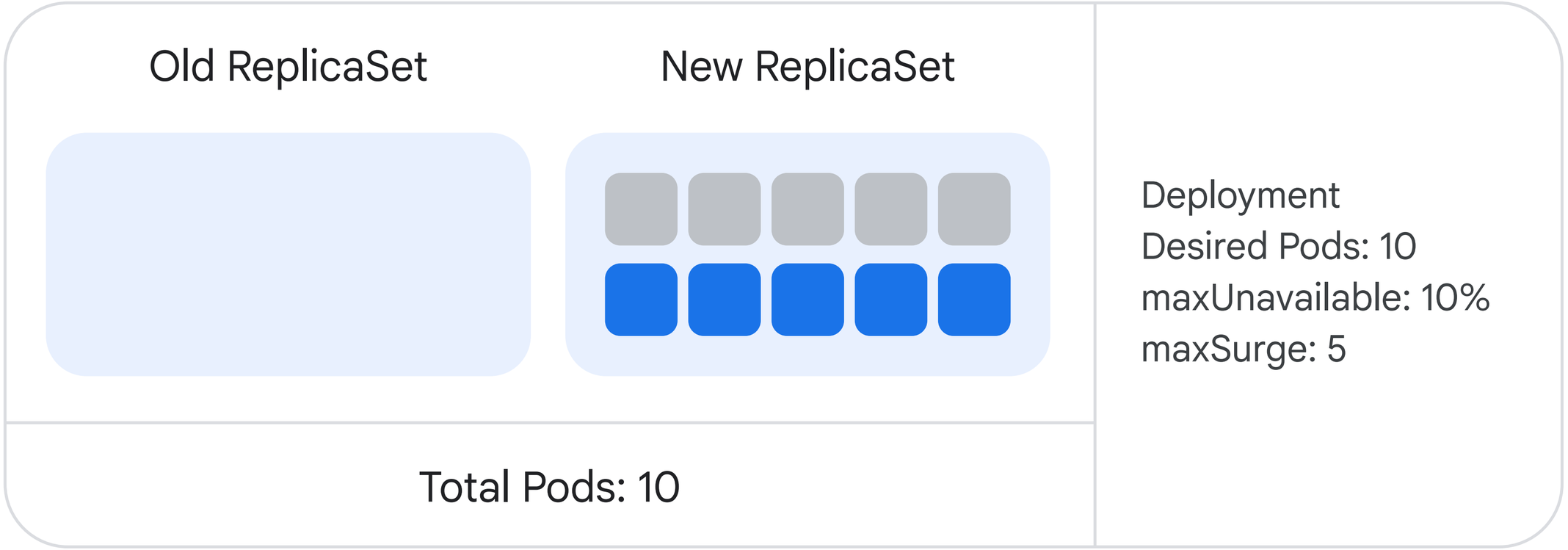
maxUnavailable



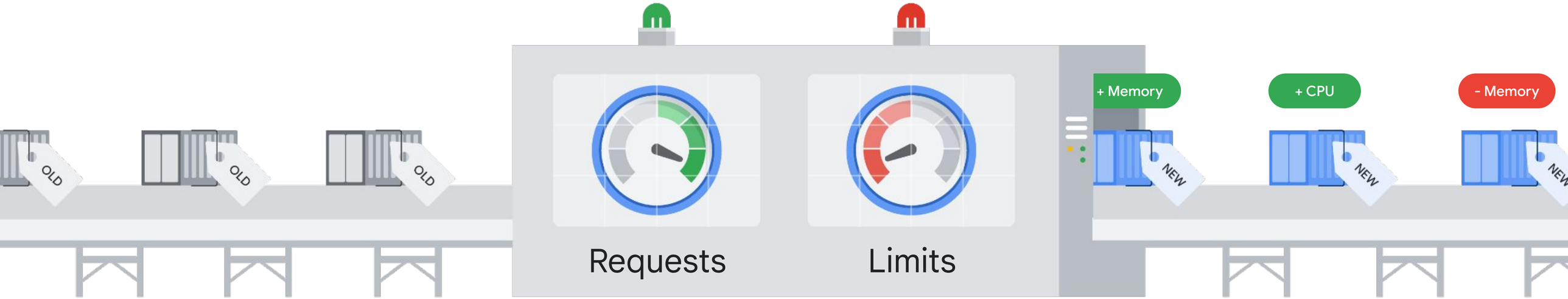
Additional Pods are launched in the new ReplicaSet



Remaining old ReplicaSet Pods are deleted



Control cluster resources with requests and limits



Can be set manually, or you can have a Pod set them for you.

Requests

Requests

Minimum amount of CPU and memory that a container will be allocated on a node.

- The scheduler will only assign a container to a node if that node has enough available resources to meet the container's requests.
- If you specify a CPU or memory value that is larger than the available resources on your nodes, your pod will never be scheduled.

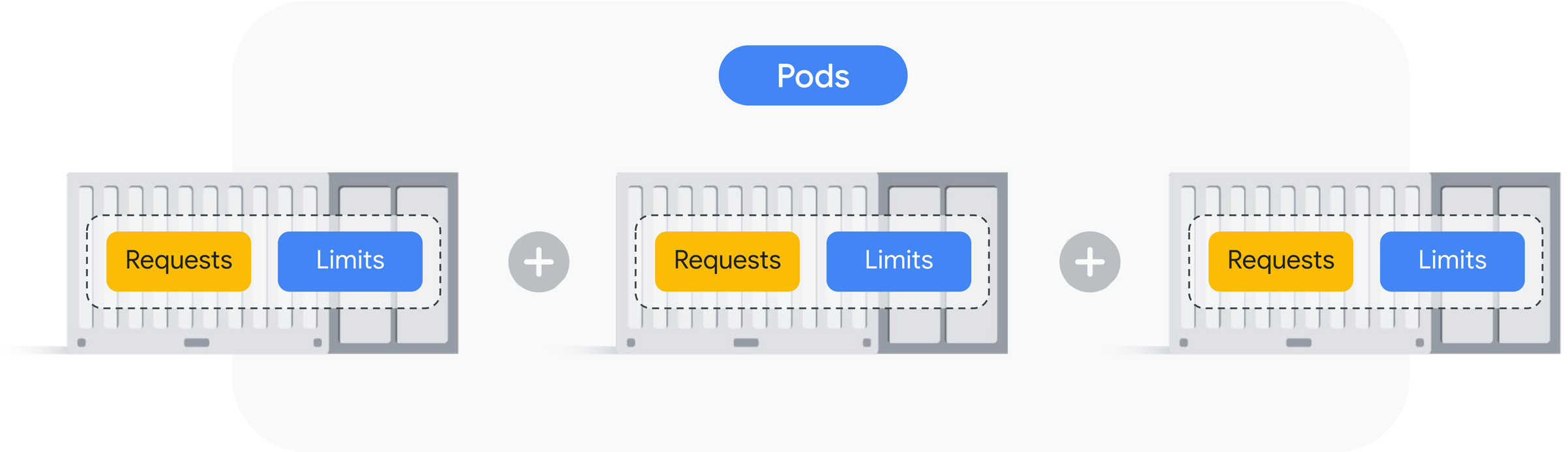
Limits

Limits

How much CPU and memory a container can use on a node.

- Prevents one container from consuming excessive resources and affecting other containers or critical node processes.
- The limit cannot be lower than the request.

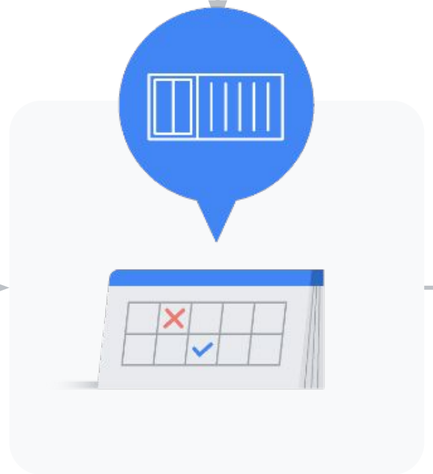
Pods are scheduled as a group



Calculate the total resource requests and limits by adding up the values from each individual container within the Pod.

Manage requests and limits

Incoming requests & limits



Kube-scheduler

Decides which node to place the Pod on.

Optimal Node?

No

Yes

Kubelet

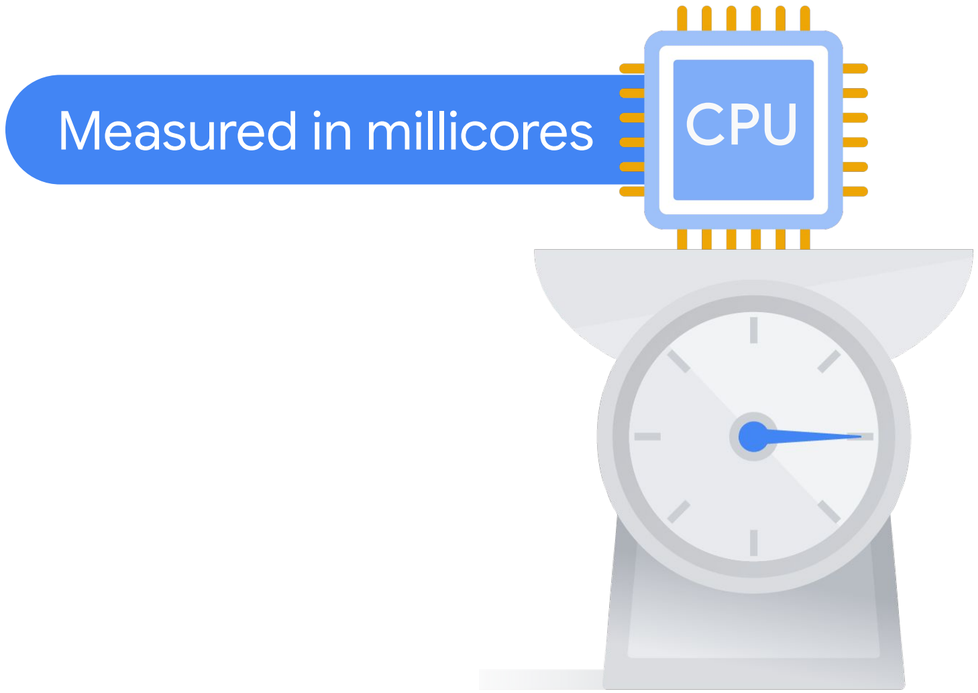
Node

Enforces resource limits.

Ensures the running container is not allowed to use more of that resource than the limits set.

Reserves request amounts for containers.

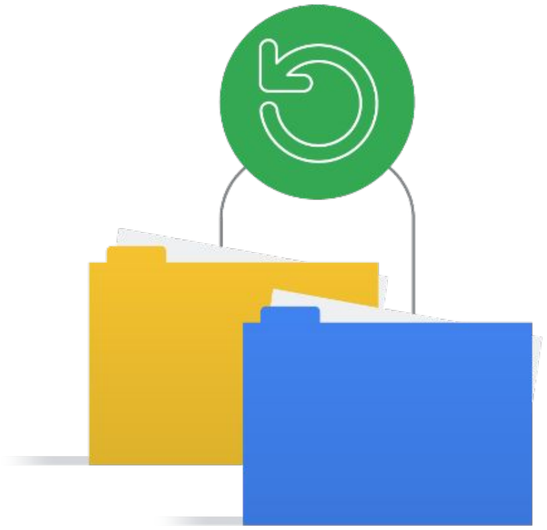
Central processing unit (CPU) resources



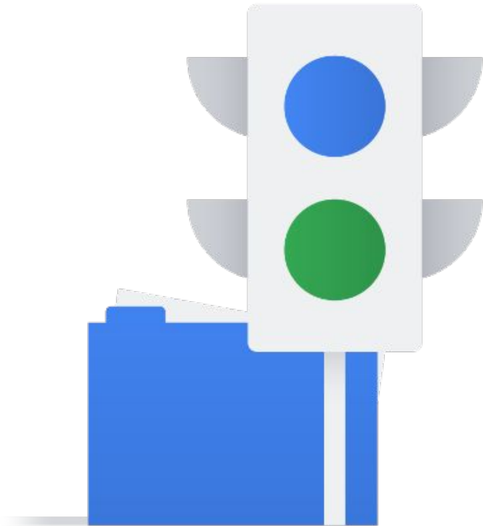
2 cores = 2000 millicores

¼ core = 250 millicores

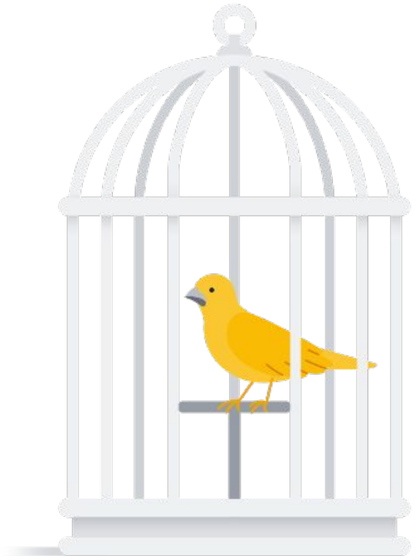
Other GKE Deployment strategies



Recreate deployment

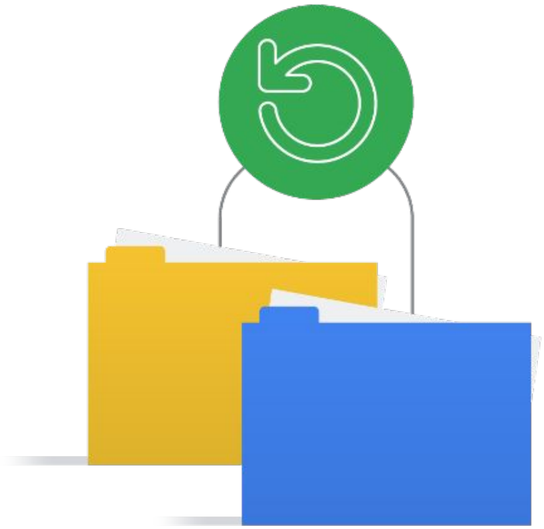


Blue/green deployment



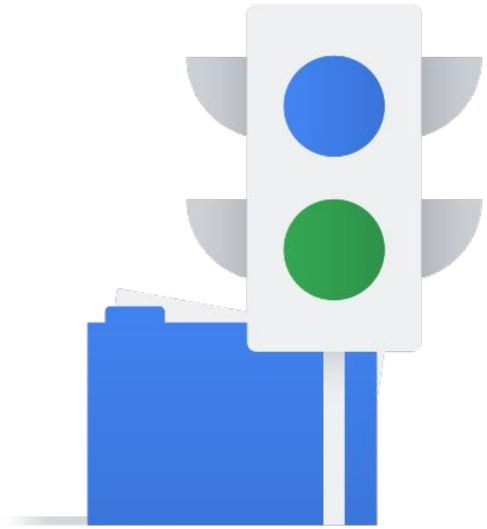
Canary test

The recreate strategy



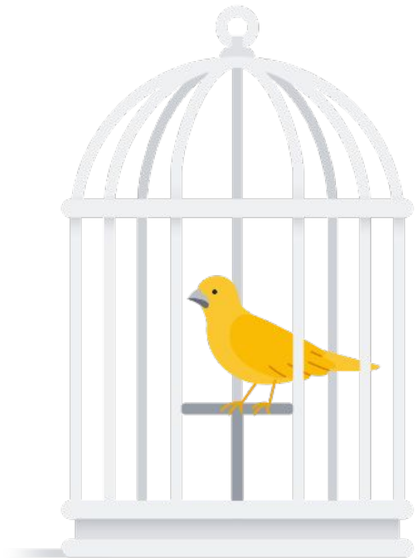
- **Strategy**: Delete old Pods before creating new ones.
- **Advantage**: All users will gain access to the updated deployment at the same time.
- **Disadvantage**: Users will experience disruptions while new Pods are being created.

The blue/green deployment strategy



- **Strategy**: Create a completely new Deployment with a newer version of the application. Blue refers to the old version, and green refers to the new version.
- **Advantage**: Rollouts are instantaneous, and the newer versions can be tested internally before releasing.
- **Disadvantage**: Resource usage is doubled during the Deployment process.

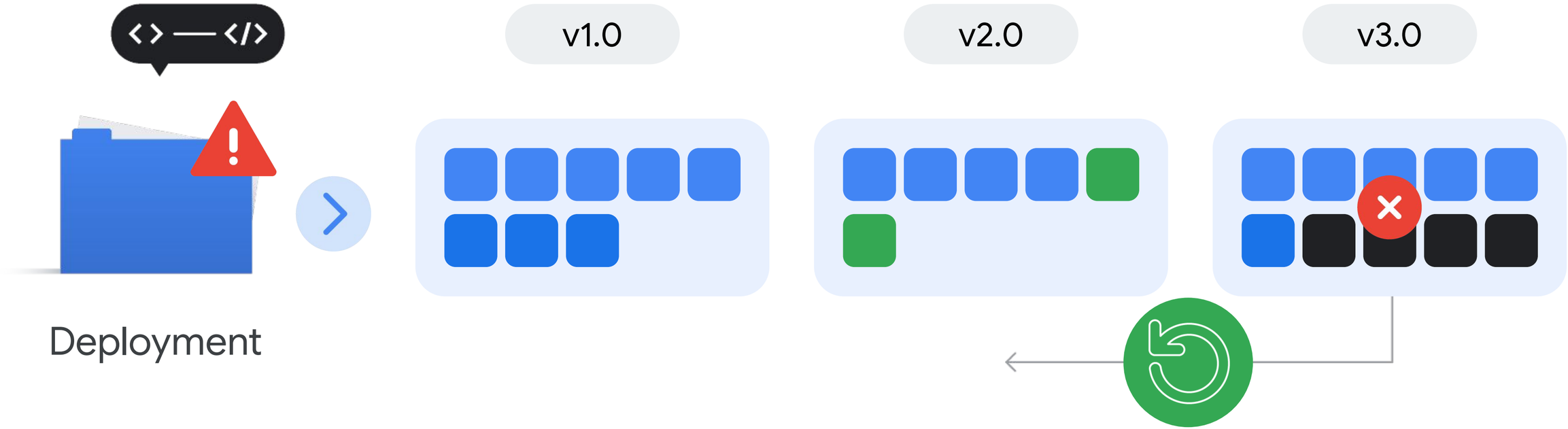
The canary strategy



- **Strategy**: Gradually move traffic to the new version of your application.
- **Advantage**: It minimizes excess resource usage during the update and helps identify issues before rollout.
- **Disadvantages**: It’s slower and may require additional tooling.

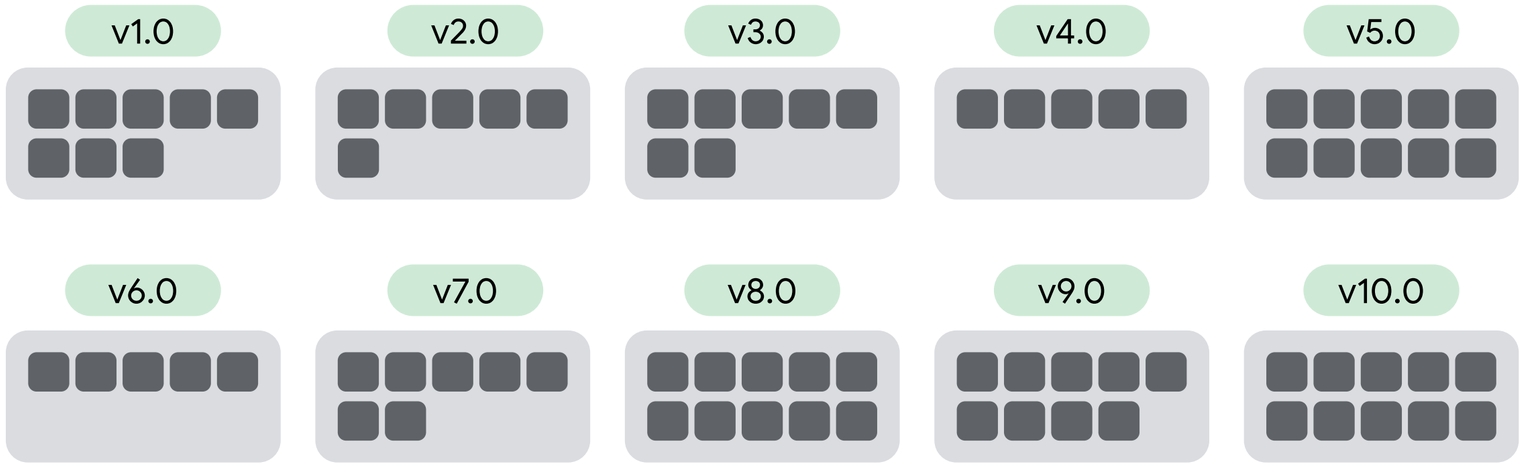
Rolling back updates

```
$ kubectl rollout undo deployment [DEPLOYMENT_NAME]
```



Inspect the rollout history

```
$ kubectl rollout history deployment [DEPLOYMENT_NAME] --revision=2
```

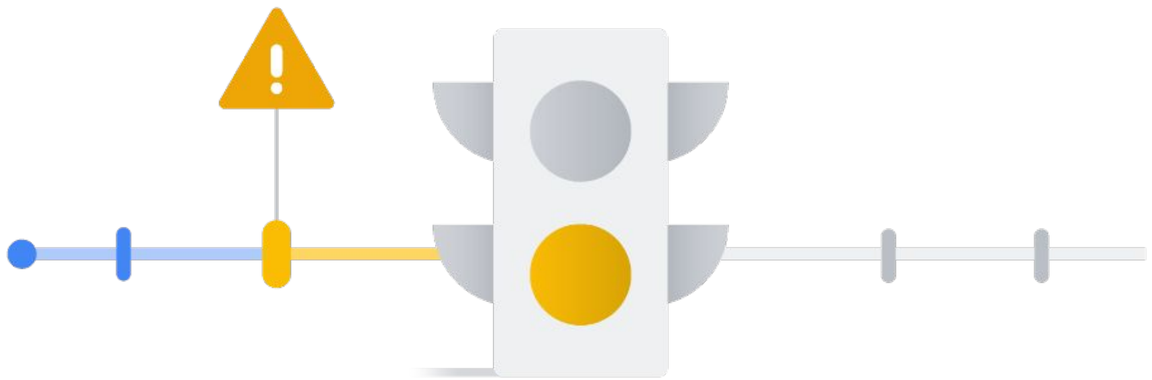


- Cloud Shell
- Google Cloud console

By default, you can inspect the rollout history from the previous 10 ReplicaSets.

Pause rollouts to troubleshoot issues

```
$ kubectl rollout pause deployment [DEPLOYMENT_NAME]
```

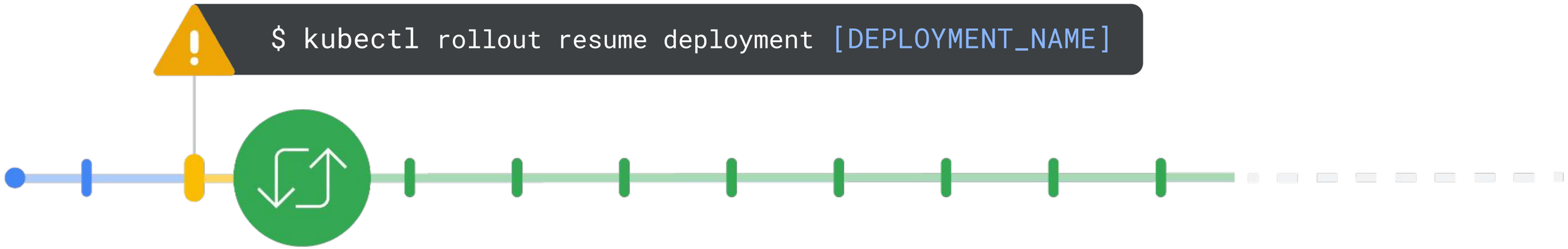


An automatic rollout is triggered when a deployment is edited.

Frequent updates lead to a large amount of rollouts.

The kubectl rollout pause command will halt a rollout so that you can troubleshoot.

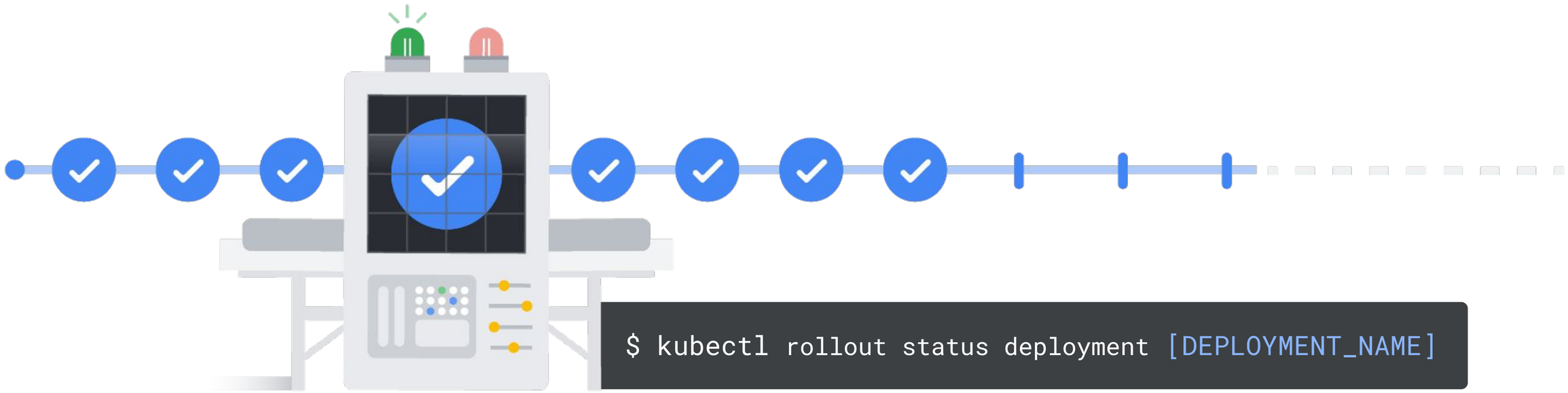
Resume rollouts



New changes to the deployment will only be implemented after the rollout is resumed.

All new changes will be released as one new version.

Monitor the status of a rollout

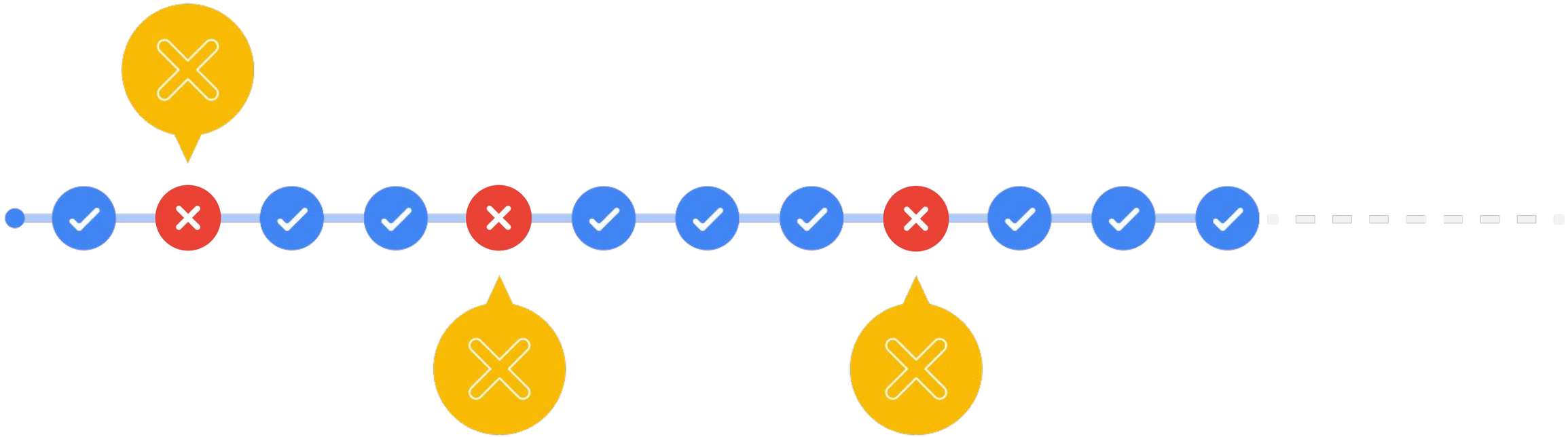


```
$ kubectl rollout status deployment [DEPLOYMENT_NAME]
```

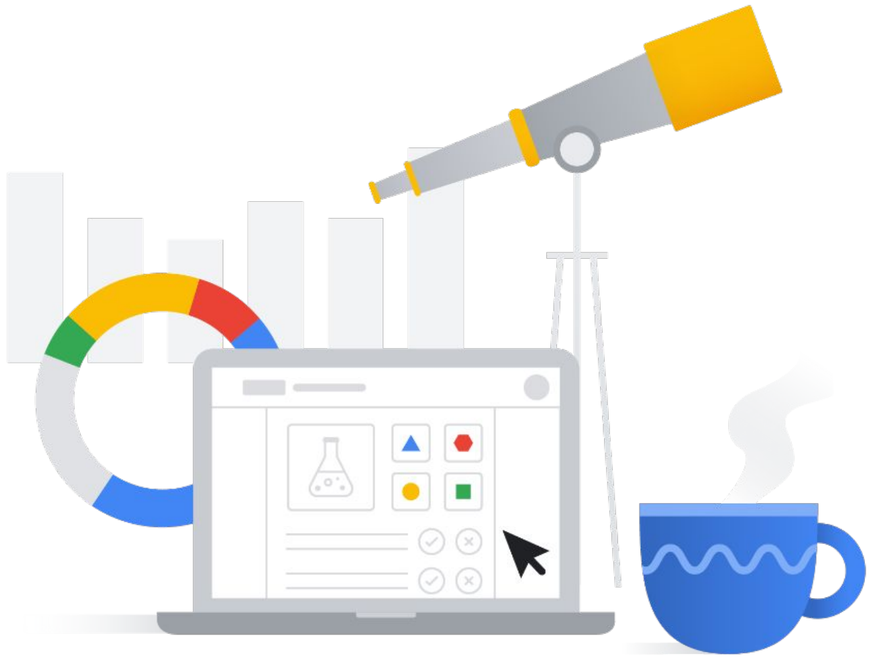
Delete a rollout

```
$ kubectl delete deployment [DEPLOYMENT_NAME]
```

Google Cloud console



Lab: Creating Google Kubernetes Engine Deployments



01 Create deployment manifests, deploy to cluster, and verify Pod rescheduling as nodes are disabled.

02 Trigger manual scaling up and down of Pods in deployments.

03 Trigger deployment rollout and rollbacks.

04 Perform a Canary deployment.

Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

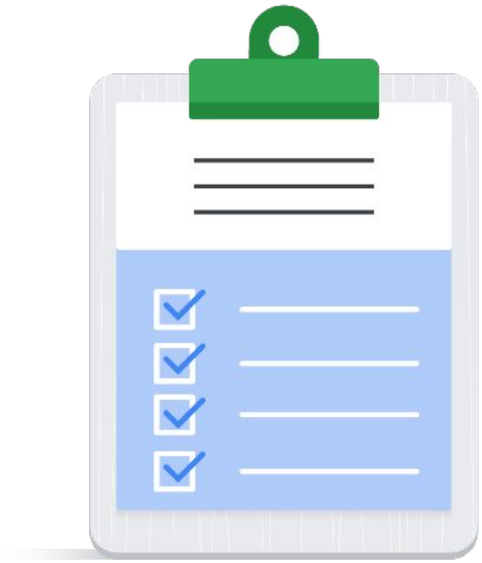
03 Scale clusters

04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

06 Get software into a cluster

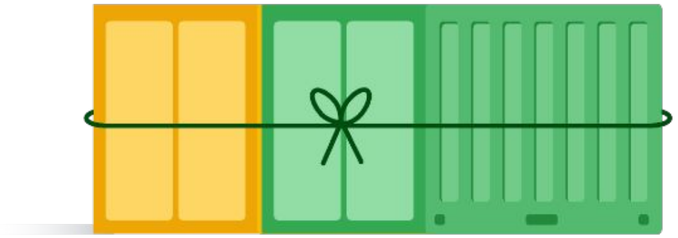
Jobs create Pods to execute tasks



Job

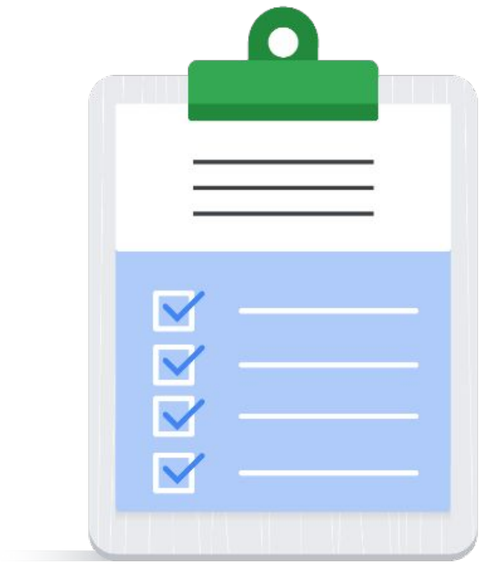
A Kubernetes resource.

A Job creates a single Pod and tracks the task to completion within that Pod.



Task

Jobs manage tasks to completion



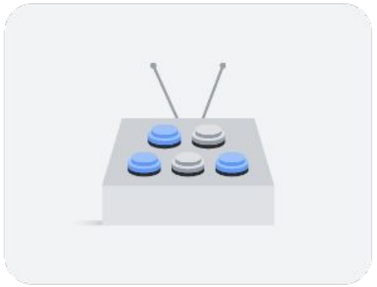
Job

manages a task to completion, rather than to a desired state

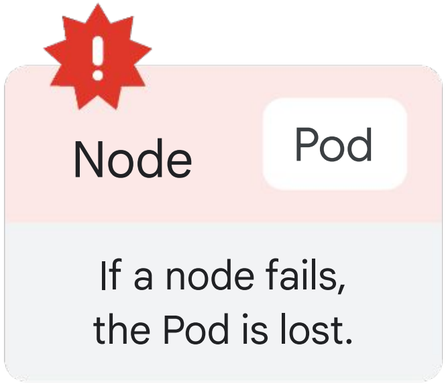
Task: Transcode

Job manifest: .yaml file

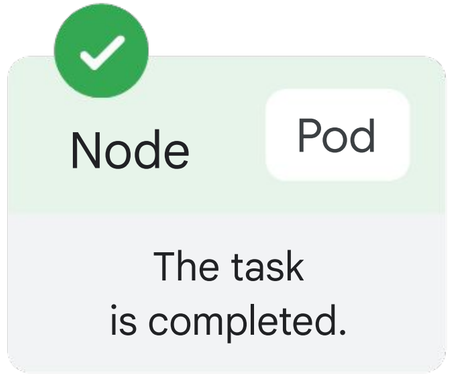
Job



Job controller

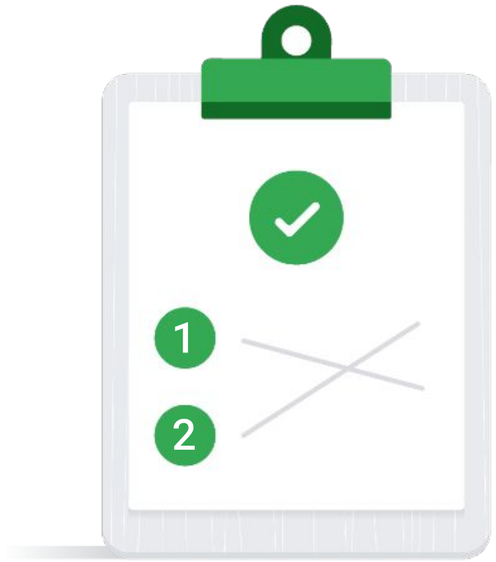


If a node fails, the Pod is lost.

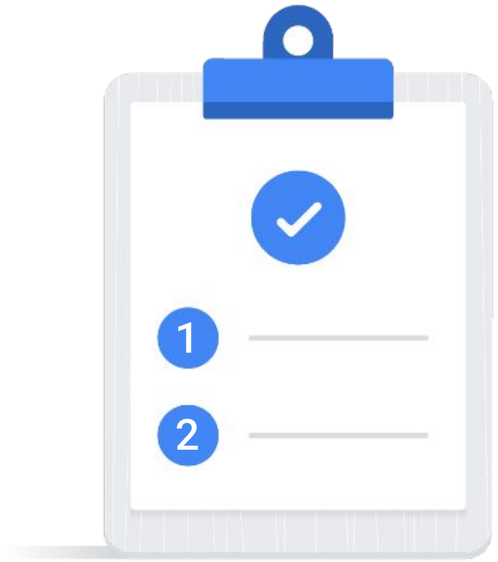


The task is completed.

There are two types of Jobs

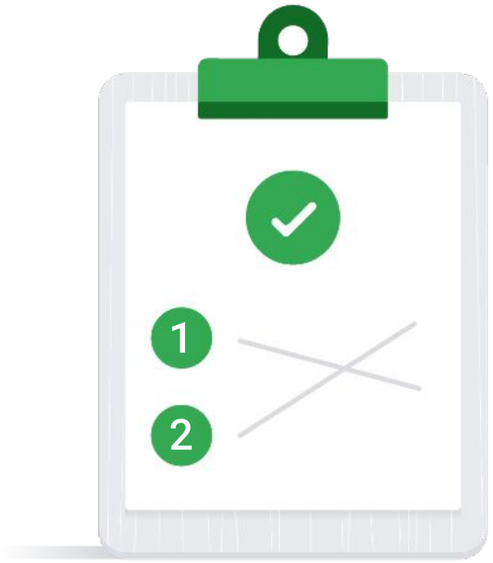


Non-parallel



Parallel

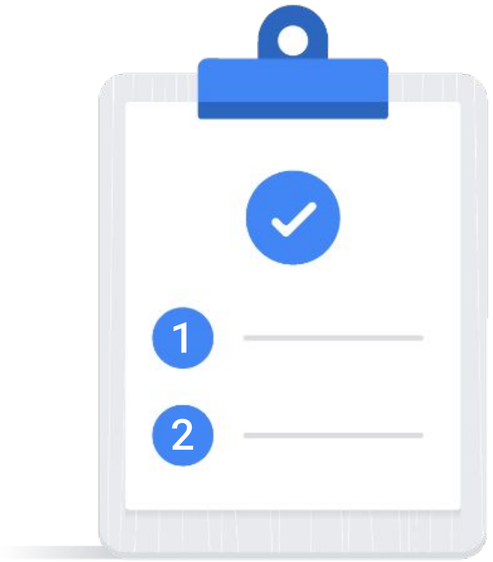
Non-parallel Jobs



Non-parallel

- Used to run a single task one time and ensure its completion.
  - Image processing
  - Data migration
- One Pod executes the task.
- The Job is finished when the Pod exits successfully, or when the required number of completions is reached.
- If the Pod fails, it will be recreated.

Parallel Jobs



Parallel

- Schedule multiple Pods to work on a Job concurrently, but with a predefined limit on the number of completions.
- Each Pod works independently on its assigned task.
- The Job finishes when the specified number of tasks are completed successfully.
- Useful when tasks need to be completed more than once:
  - Bulk image resizing
  - Parallel scientific computations

The Kind property

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

```
$ kubectl apply –f [JOB_FILE]
```

```
$ kubectl run pi --image perl:5.34 --restart Never -- perl
-Mbignum=bpi -wle 'print bpi(2000)'
```

Acts as a label that tells Kubernetes:

- What type of batch process the Job represents.
- How it should be handled.

The Job spec and Pod template

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Job Spec: How the Job should perform its tasks.

Pod template: A blueprint for the Pods that the Job will create.

```
$ kubectl apply –f [JOB_FILE]
```

```
$ kubectl run pi --image perl:5.34 --restart Never -- perl
-Mbignum=bpi -wle 'print bpi(2000)'
```

Configuring the restartPolicy

1. `restartPolicy: Never`

   Any container failure within a Pod will cause the entire Pod to fail permanently.

2. `restartPolicy: onFailure`

   The Pod remains on the node, but the Container restarts.

backOffLimit field controls restarts

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl:5.34
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

Controls how many times the Job controller should attempt to restart failed Pods.

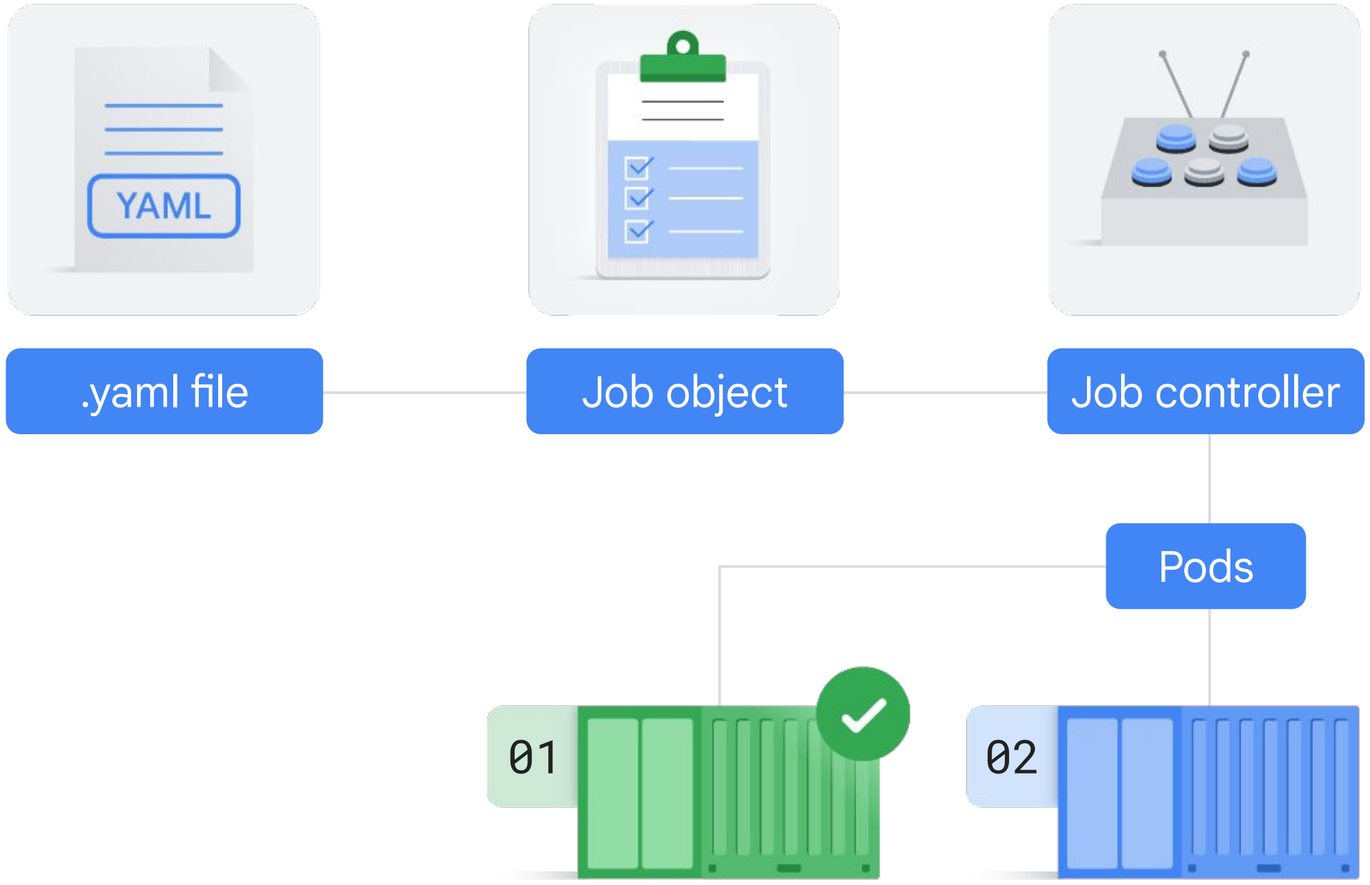
```
$ kubectl apply –f [JOB_FILE]
```

```
$ kubectl run pi --image perl:5.34 --restart Never -- perl
-Mbignum=bpi -wle 'print bpi(2000)'
```

Identify completions and parallelism

Parallel Job with fixed completion count

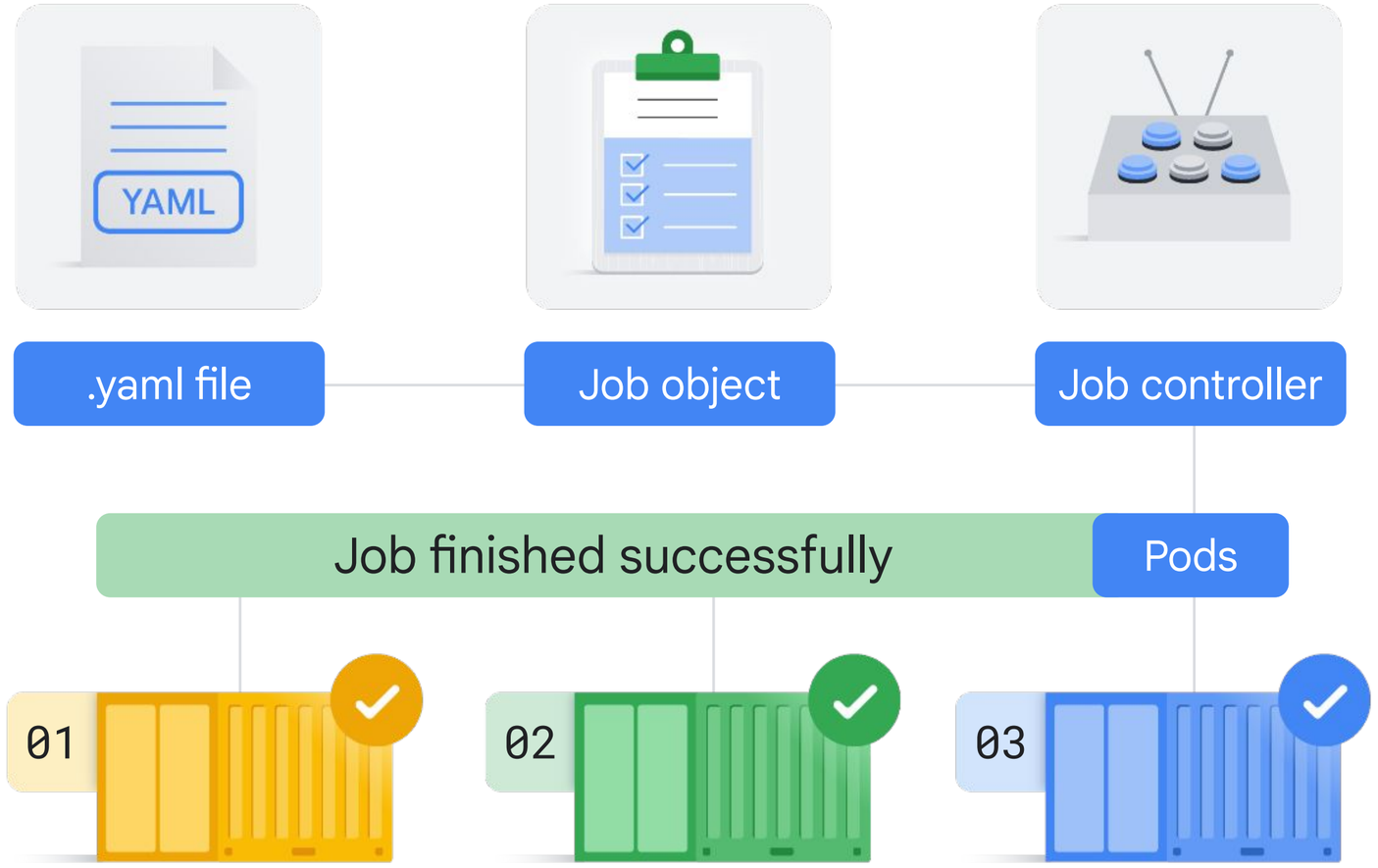
```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-app-job
spec:
  completions: 3
  parallelism: 2
  template:
    spec:
[…]
```



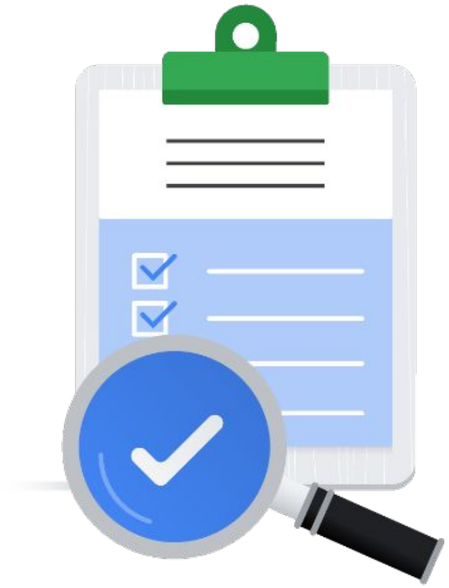
New Pods launch until completion

Parallel Job with fixed completion count

```
apiVersion: batch/v1
kind: Job
metadata:
  name: my-app-job
spec:
  completions: 3
  parallelism: 2
  template:
    spec:
[…]
```



Inspecting Jobs and filtering pods



```
$ kubectl describe
job [JOB_NAME]
```

Used to inspect Jobs.

```
$ kubectl get pod -l
[job-name=my-app-job]
```

Used to filter Pods.

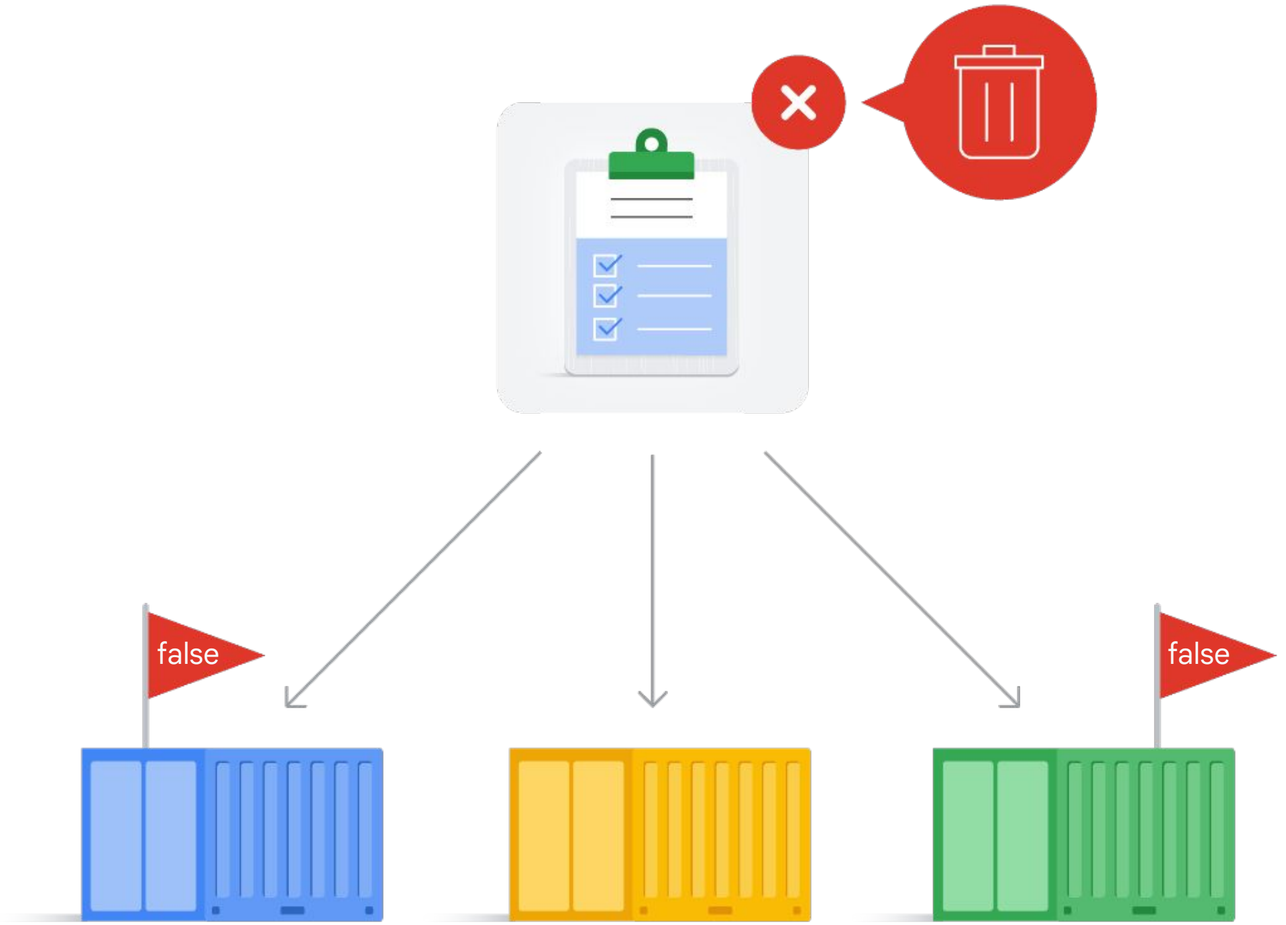
Delete a Job

```
$ kubectl delete -f [JOB_FILE]
```

```
$ kubectl delete job [JOB_NAME]
```

```
$ kubectl delete job [JOB_NAME]
   --cascade false
```

Delete jobs directly in the Google Cloud console



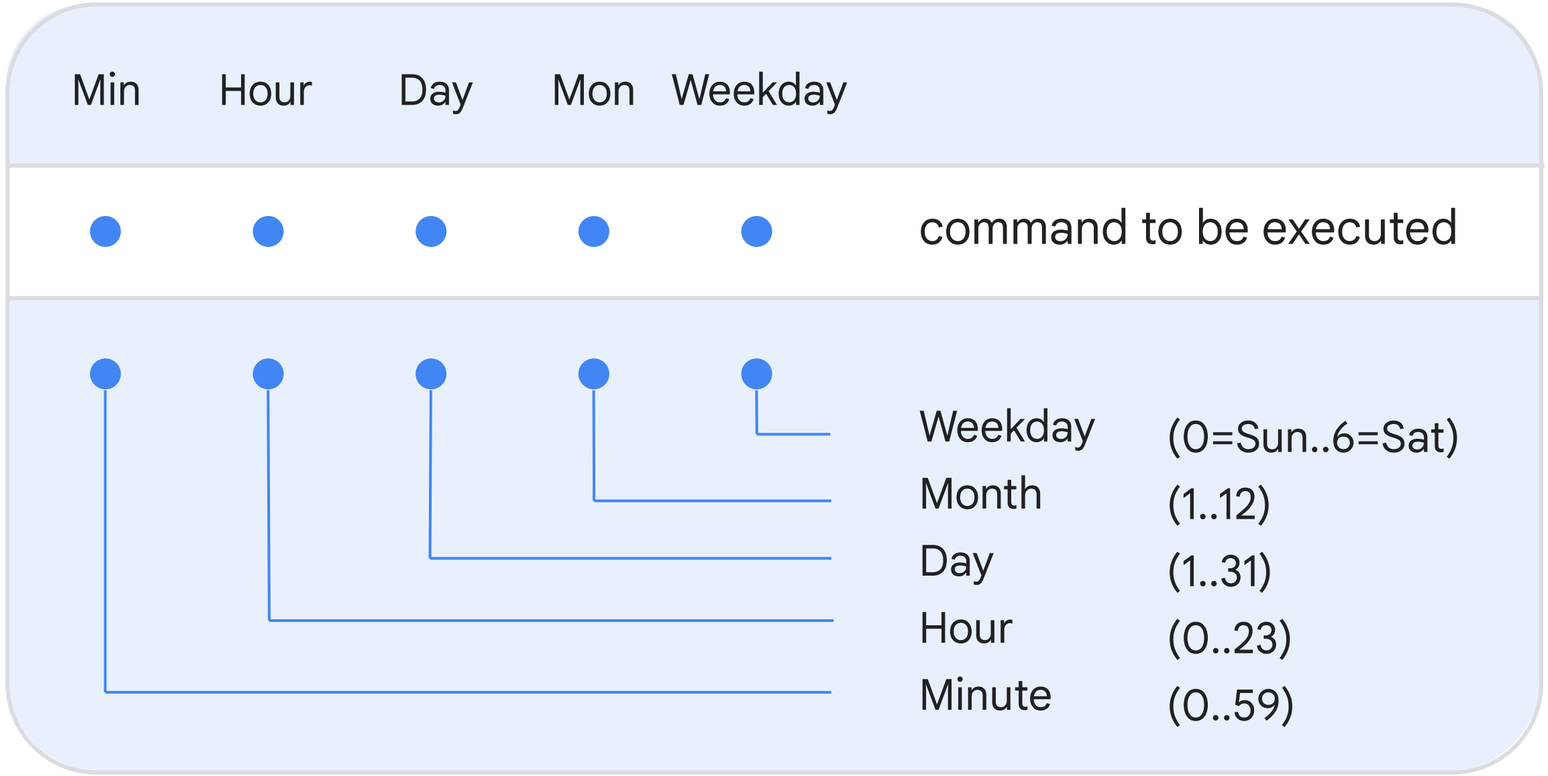
Use CronJobs for repeated execution

CronJobs

Specify precise dates and times for repeated execution

Cron uses an easy-to-read text format

- Different parts separated by spaces.
- Each part controls a different aspect of when and what
to execute.
- Use an * asterisk to represent an entire range of possible values.



Specify individual times or ranges

Examples

`0 * * * *`	Every hour
`*/15 * * * *`	Every 15 minutes
`0 */6 * * *`	Every 6 hours
`0 20 * * 1-5`	Every week Mon-Fri at 8 pm
`0 0 1 */2 *`	At 00:00 on day 1 of every other month

Schedule Jobs by specifying individual times or ranges.

Repeat values at regular intervals

Examples	
`0 * * * *`	Every hour
`*/15 * * * *`	Every 15 minutes
`0 */6 * * *`	Every 6 hours
`0 20 * * 1-5`	Every week Mon-Fri at 8 pm
`0 0 1 */2 *`	At 00:00 on day 1 of every other month

To make a value repeat at regular intervals, add a slash followed by the interval number.

Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

03 Scale clusters

04 Control Pod placement with labels and affinity rules

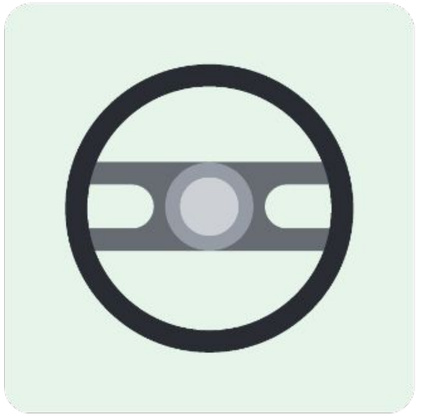
05 Control Pod placement with taints and tolerations

06 Get software into a cluster

Use GKE Standard mode for manual cluster scaling

Standard mode

Custom-managed cluster scaling

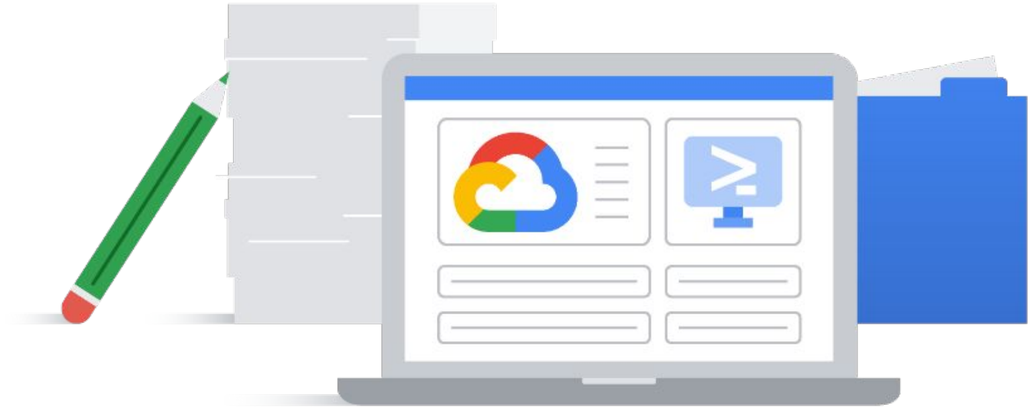


Autopilot mode

Automatic cluster scaling

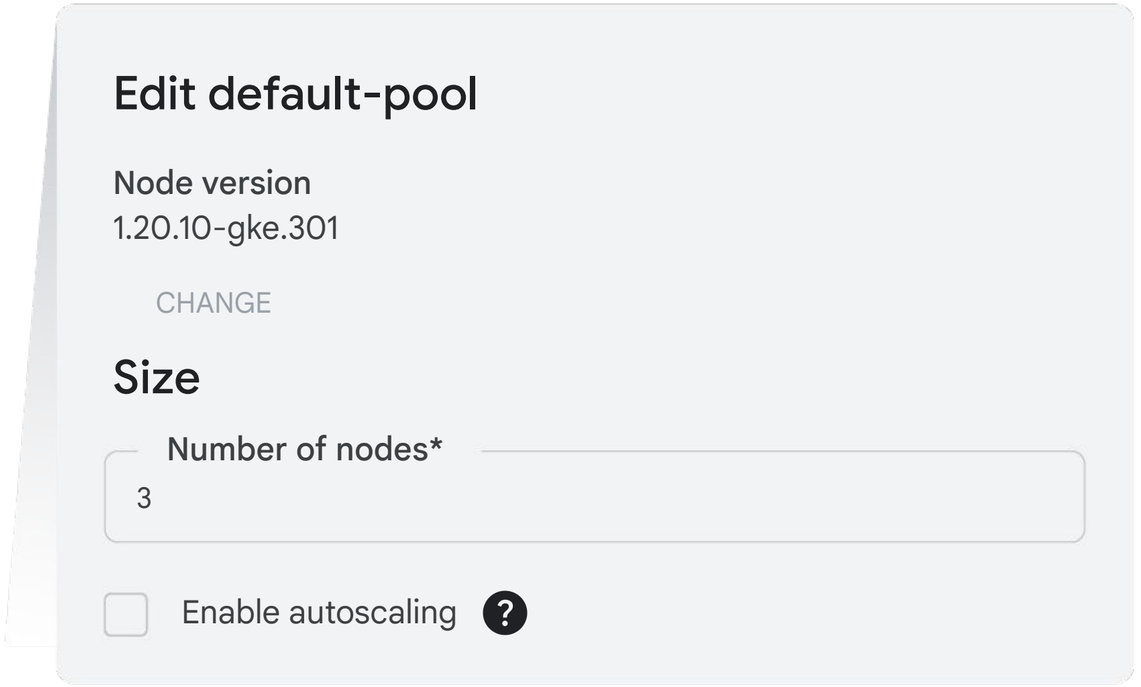
Application resources fluctuate

Easily scale a cluster up or down to match those changing demands.



- Google Cloud console
- Cloud Shell

Node pools



1. A node pool **groups nodes** that have the same configuration type within a cluster.
2. NodeConfig specification
3. Container cluster

A cluster can not be entirely shut down

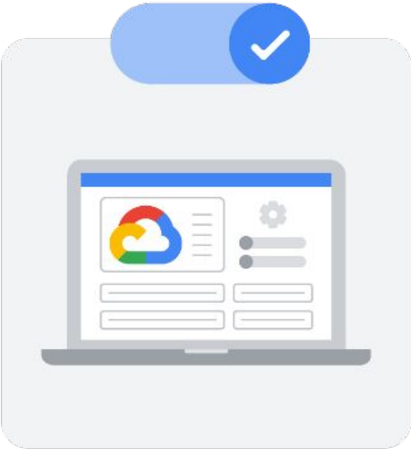
- A cluster must have at least one node to run system Pods.
- A cluster can be scaled down to 0, but the cluster itself can not be entirely shut down.

The resize gcloud command for manual scaling

Standard mode

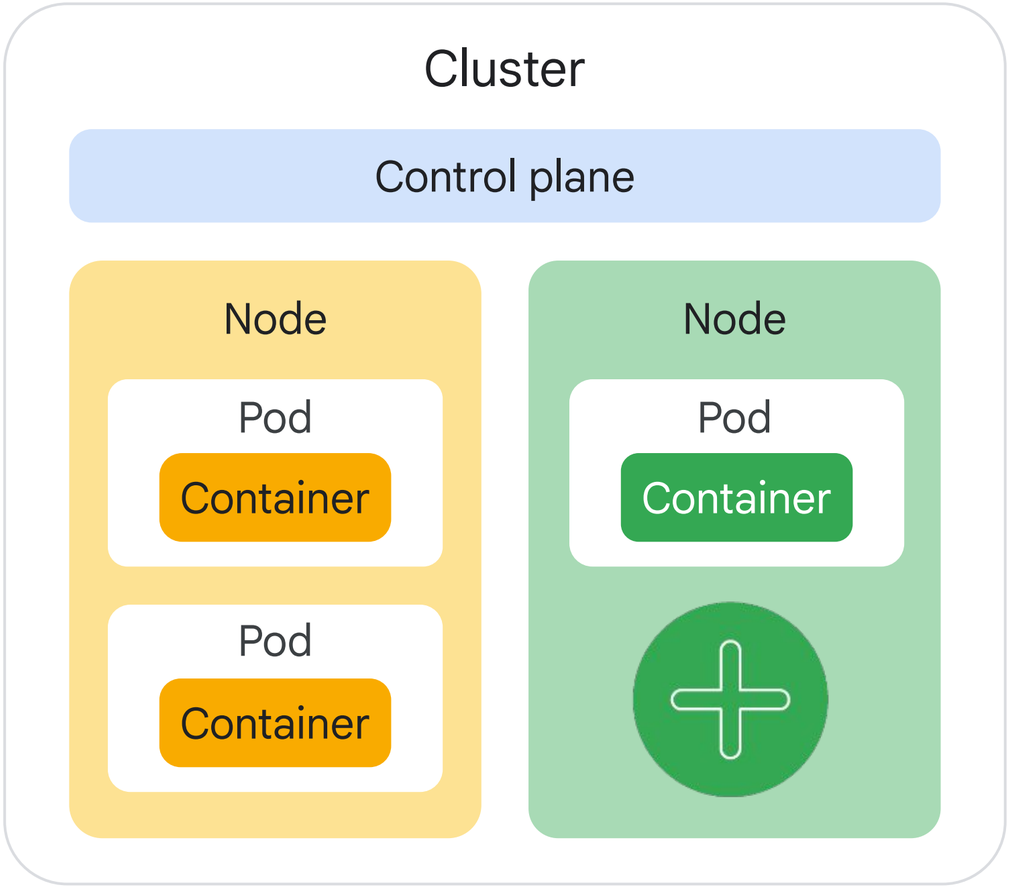
```
gcloud container clusters resize projectdemo
--node-pool default-pool \
  --size 6
```

- Manually resizes a cluster.
- Removes instances at random.
- Terminates running Pods gracefully.



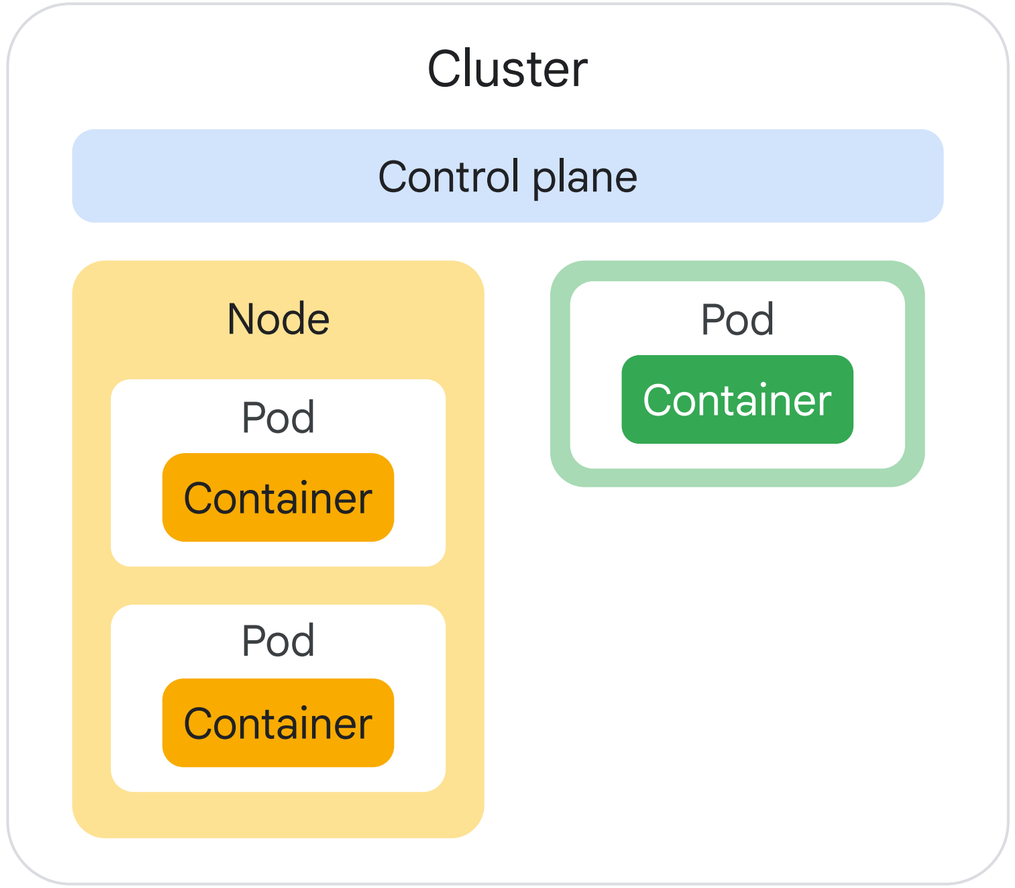
Google Cloud console

GKE cluster autoscaler



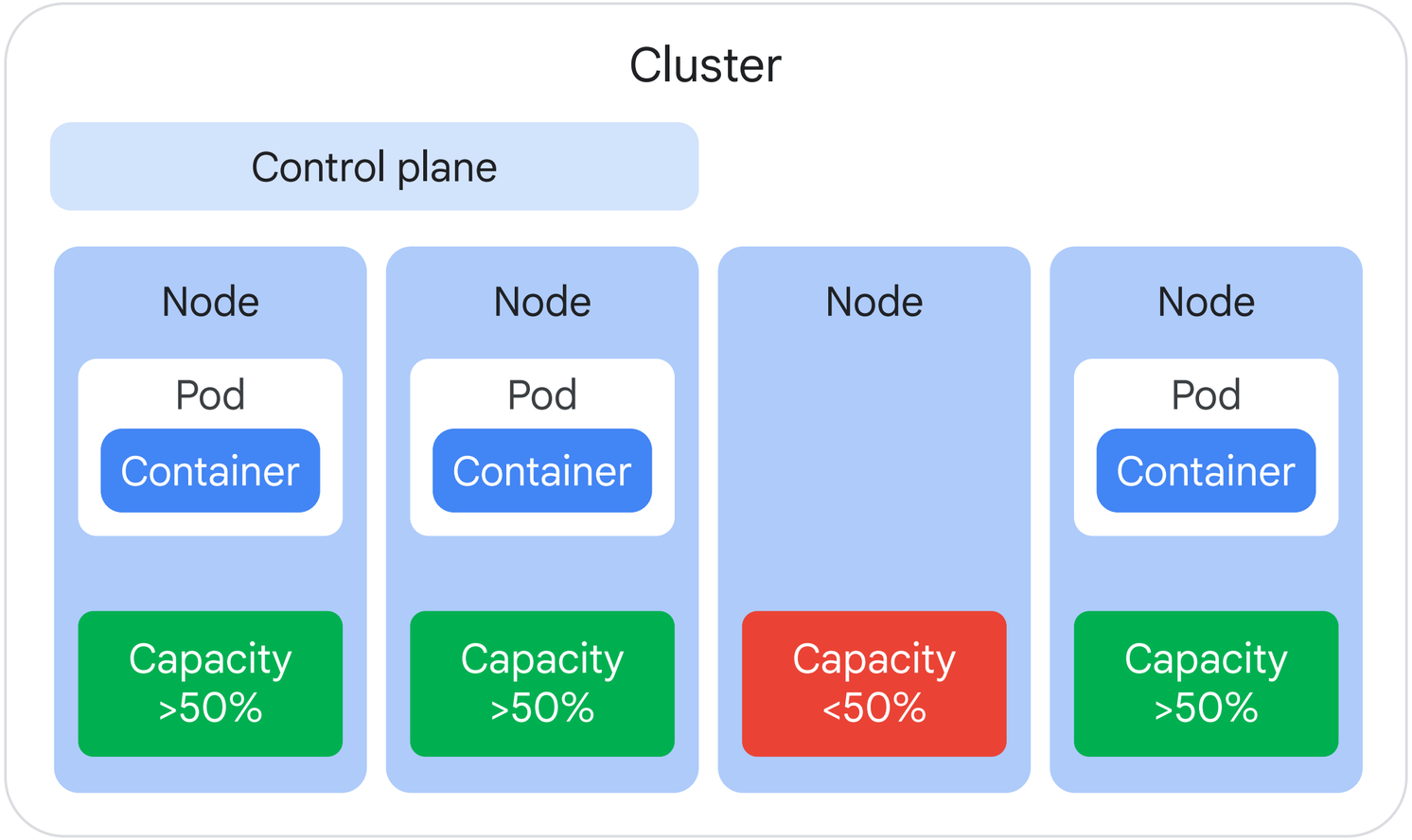
- Available on Standard cluster node pools.
- Automatically resizes a cluster based on the workloads' resource demands.
- Disabled by default.
- Automatically adds new nodes to a cluster.
- Deletes underutilized nodes.

Nodes with low resource capacity



- Scenario: all of your node pools are low on resource capacity.
- Solution: terminate Pods or add more nodes.
- Process:
  - Pod enters a holding pattern.
  - The scheduler sets the schedulable Pod condition to false and marks it as `unschedulable`.

Cluster autoscaler checks for disposable nodes



A node is considered disposable if all are true:

- Total CPU and memory is less than 50% of a node's allocatable capacity.
- All pods running on the node can be moved to other nodes.
- The cluster does not have scale-down disabled.

Cluster autoscaler limits

Cluster scaler can handle up to

15k

nodes

Each node supports a maximum of

256

Pods

Cluster-wide limit of

200k

Pods

gcloud commands for autoscaling

Create a cluster with autoscaling enabled

```
gcloud container clusters create
[CLUSTER_NAME] --enable-autoscaling
--min-nodes 15 --max-nodes 50 [--zone
COMPUTE_ZONE]
```

Enable autoscaling for an existing node pool

```
gcloud container clusters update
[CLUSTER_NAME] --enable-autoscaling \
--min-nodes 1 --max-nodes 10 --zone
[COMPUTE_ZONE] --node-pool [POOL_NAME]
```

Add a node pool with autoscaling enabled

```
gcloud container node-pools create
[POOL_NAME] --cluster [CLUSTER_NAME]
--enable-autoscaling --min-nodes 15
--max-nodes 50 [--zone COMPUTE_ZONE]
```

Disable autoscaling for an existing node pool

```
gcloud container clusters update
[CLUSTER_NAME] --no-enable-autoscaling \
--node-pool [POOL_NAME] [--zone
[COMPUTE_ZONE] --project [PROJECT_ID]]
```

Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

03 Scale clusters

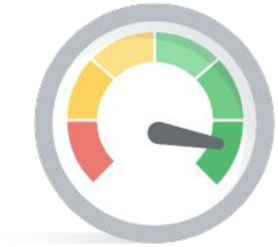
04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

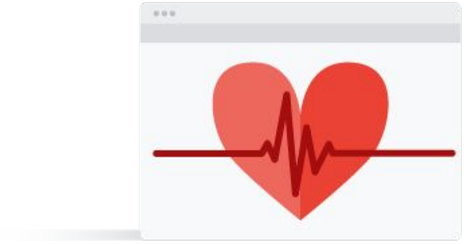
06 Get software into a cluster

Pod placement in GKE

The process of controlling where your application's Pods are deployed across the cluster's nodes.



Optimize performance



Ensure availability

Manage resource allocation

Control and manage Pod placement

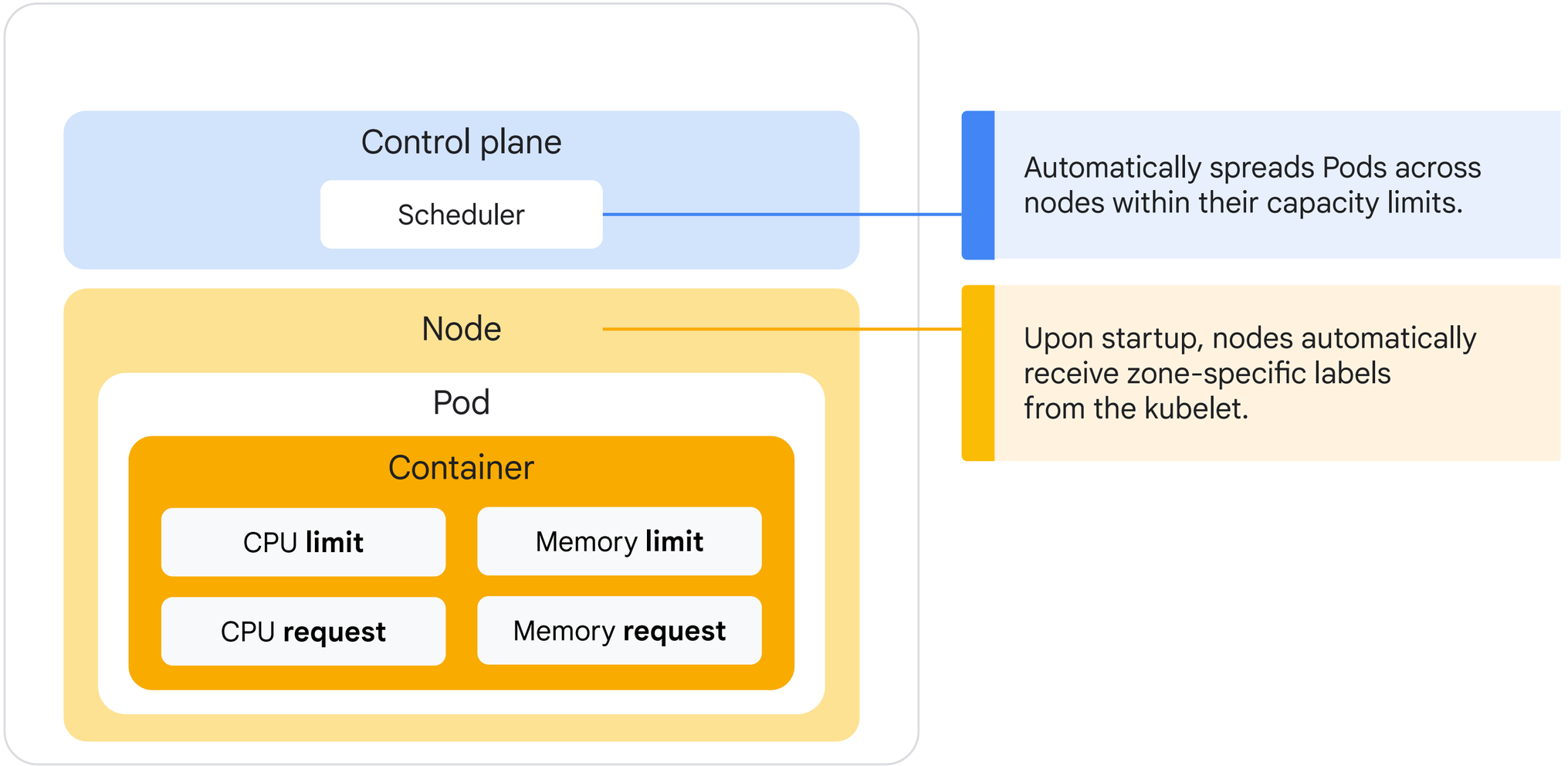
Labels

Affinity rules

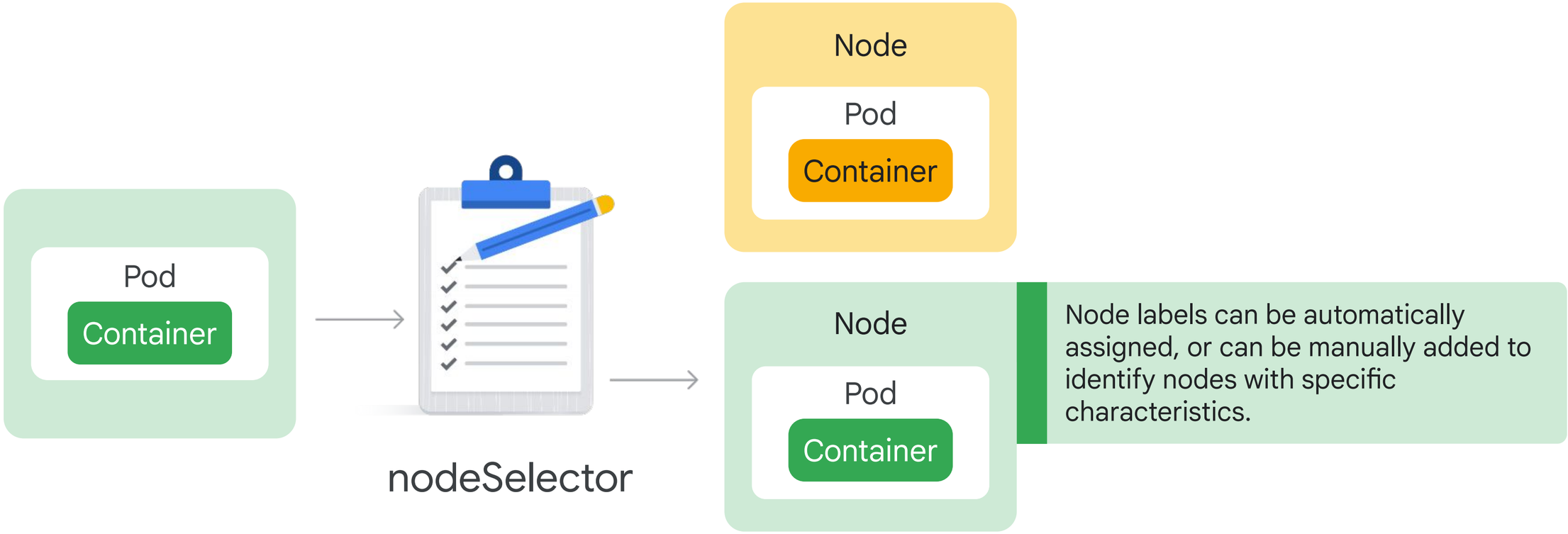
Taints

Tolerations

Controlled scheduling



nodeSelector specifies preferred node labels



Match Autopilot compute class to Pod preferences

```
apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: hello-app
  spec:
   replicas: 3
    selector:
      matchLabels:
        app: hello-app
      spec:
        nodeSelector:
          cloud.google.com/compute-class: Balanced
   […]
```

Use the **nodeSelector** field to request the "Balanced" compute class and ensure Pods are placed on nodes with the resources they need.

Enable Autopilot to this by adding the **cloud.google.com/compute-class** label.

A node’s capabilities must match a Pod’s preferences

```
apiVersion: v1
kind: Pod
metadata:
  name: mysql
  labels:
   env: test
spec:
  containers:
  - name: mysql
    image: mysql
    imagePullPolicy: IfNotPresent
  nodeSelector:
    disktype: ssd
[...]
```

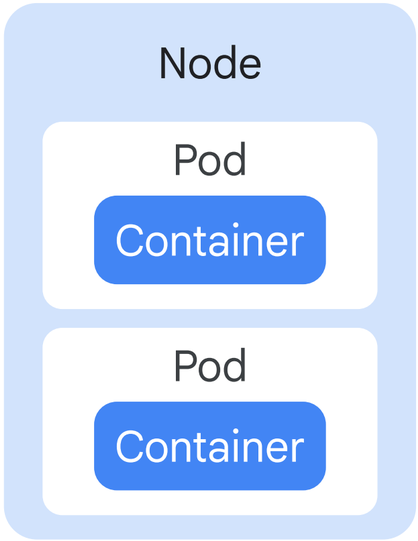
```
apiVersion: v1
kind: Node
metadata:
  name: node1
  labels:
    disktype: ssd
[…]
```

If a node's capabilities match the Pod’s preferences, then the node is considered a suitable host for that Pod.

Affinity rules influence Pod placement

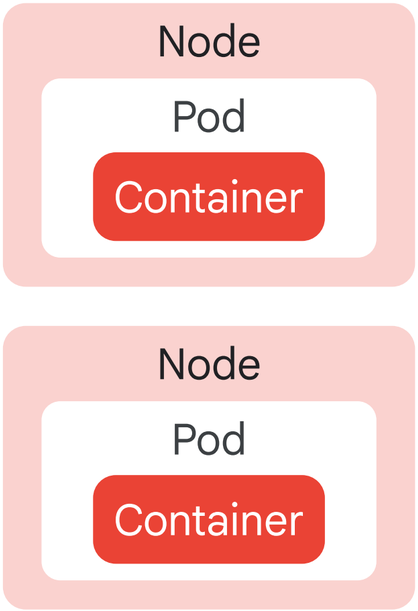
Affinity

Used to ensure that two or more Pods are placed together on the same node.



Anti-affinity

Used to spread Pods across nodes.



Affinity and anti-affinity rules keywords

01	`requiredDuringScheduling IgnoredDuringExecution`	Enforces a strict requirement that must be met when scheduling Pods.
02	`preferredDuringScheduling IgnoredDuringExecution`	Indicates a flexible preference that isn’t mandatory.

Preference weights

`preferredDuringSchedulingIgnoredDuringExecution`

Indicates that changes to labels will not affect Pods that are already running.

Weakest

Strongest

The stronger the weight, the more the scheduler will try to favor nodes that match your preference.

Scenario 1

The NodeSelectorTerms field is used and the node must meet all matchExpression requirements.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    nodeSelectorTerms:
    - matchExpressions:
      - key: accelerator-type
        operator: In
        values:
        - gpu
        - tpu
```

Example of affinity: a single `matchExpression` `–accelerator-type–` is specified.

`In` operator indicates only one of the listed values must match the preferences.

Scenario 2

Defining the intensity of preference.

```
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 90
    preference:
    - matchExpressions:
      - key: cloud.google.com/gke-nodepool
        operator: In
        values:
        - n1-highmem-4
        - n1-highmem-8
```

A weight of 90 signals a strong preference.

When the scheduler assigns the Pod to the node, it takes this preference and weight into account.

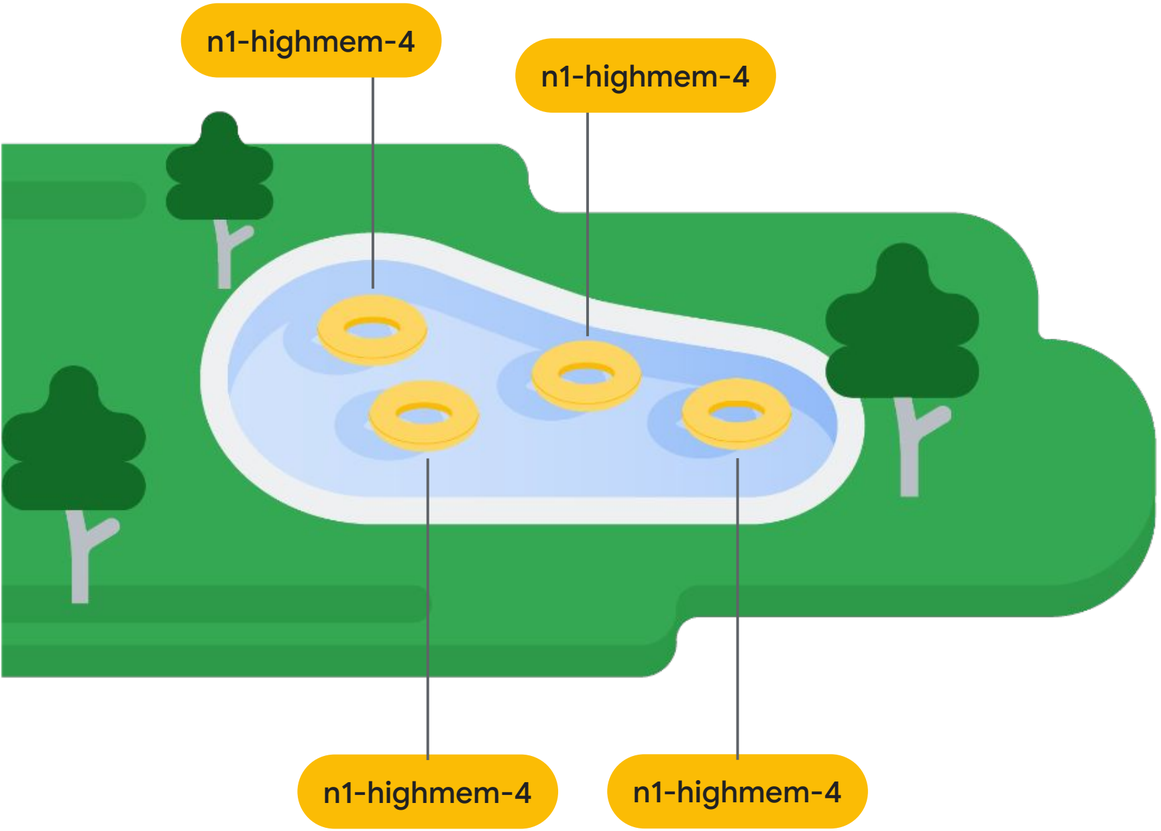
Node pool names

`n1-highmem-4`

Node pool names should indicate the type of compute instances that will be used to create the nodes.

- Descriptive
- Reflects hardware specs

GKE assigns labels based on node pool names



`n1-highmem-4`

Easily create preferences for specific hardware types using those labels during Pod scheduling.

Affinity and anti-affinity rules applications

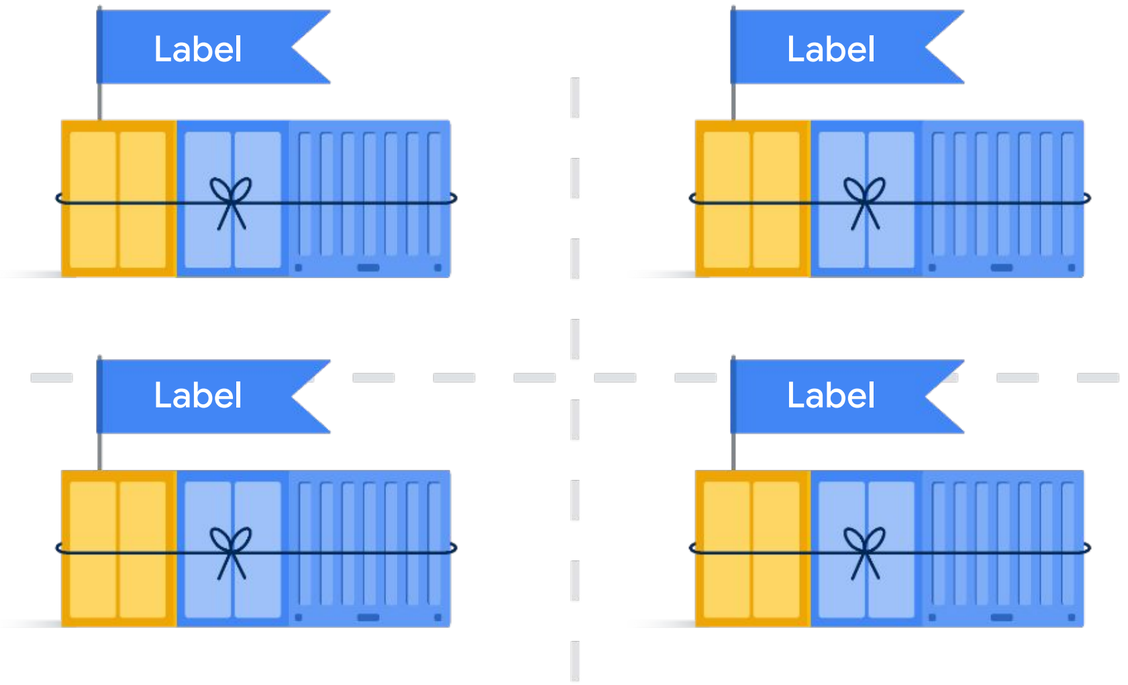
```
[...]
metadata:
  name: with-pod-affinity
spec:
  affinity:
    podAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
    podAffinityTerm:
    labelSelector:
    matchExpressions:
      - key: app
        operator: In
        values:
        - webserver
    topologyKey: topology.kubernetes.io/zone
```

Affinity and anti-affinity rules:

- Aren't limited to individual nodes.
- Can be applied at broad levels of infrastructure organization.

`topologyKeys` let you define preferences based on topology domains (e.g. zones and regions).

Granular control with inter-pod affinity and anti-affinity



Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

03 Scale clusters

04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

06 Get software into a cluster

Taints and tolerations

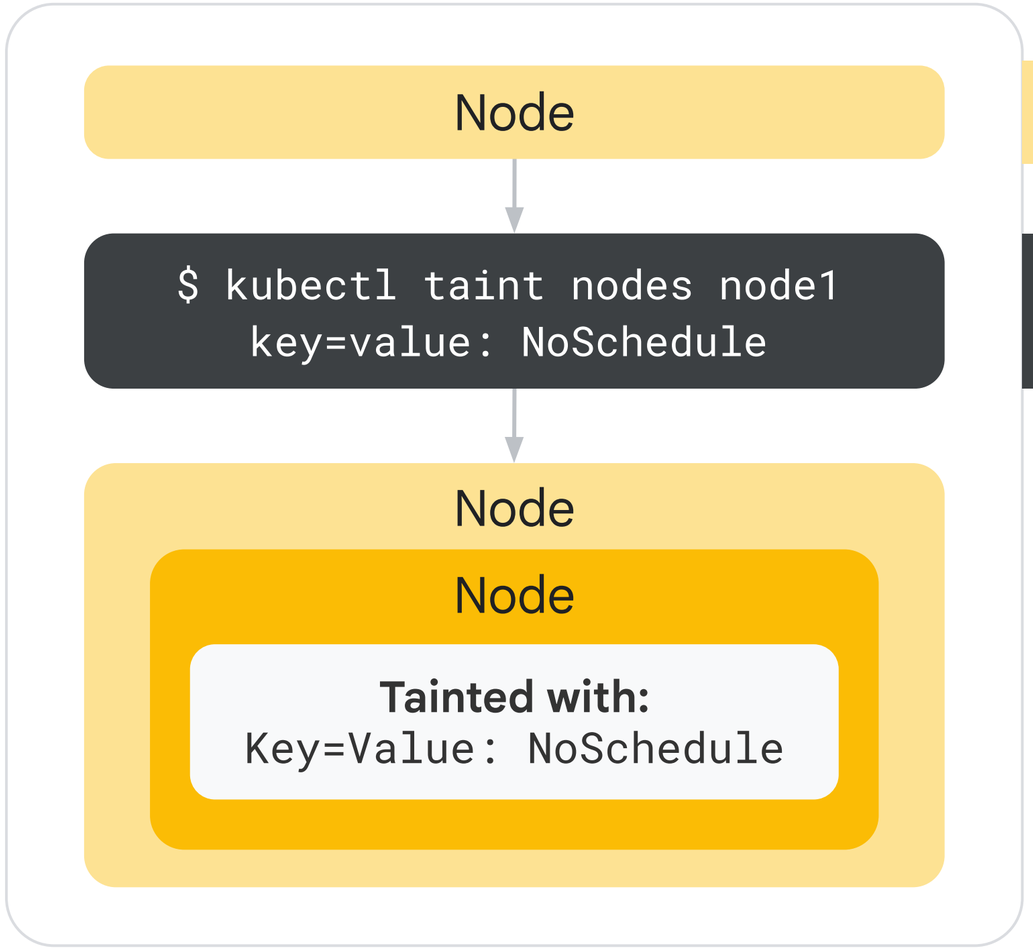
Taint

A special label that acts as a restriction indicating node suitability for a Pod.

Toleration

A specification that allows the Pod to be scheduled onto a node that has a matching taint.

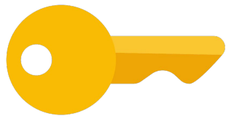
Taints are configured on nodes



Taints affect all Pods in the cluster.

Apply a taint on a node by using the `kubectl taint` command.

A taint consists of three elements



Key

Gives a descriptive name that represents the taint's purpose.

Value

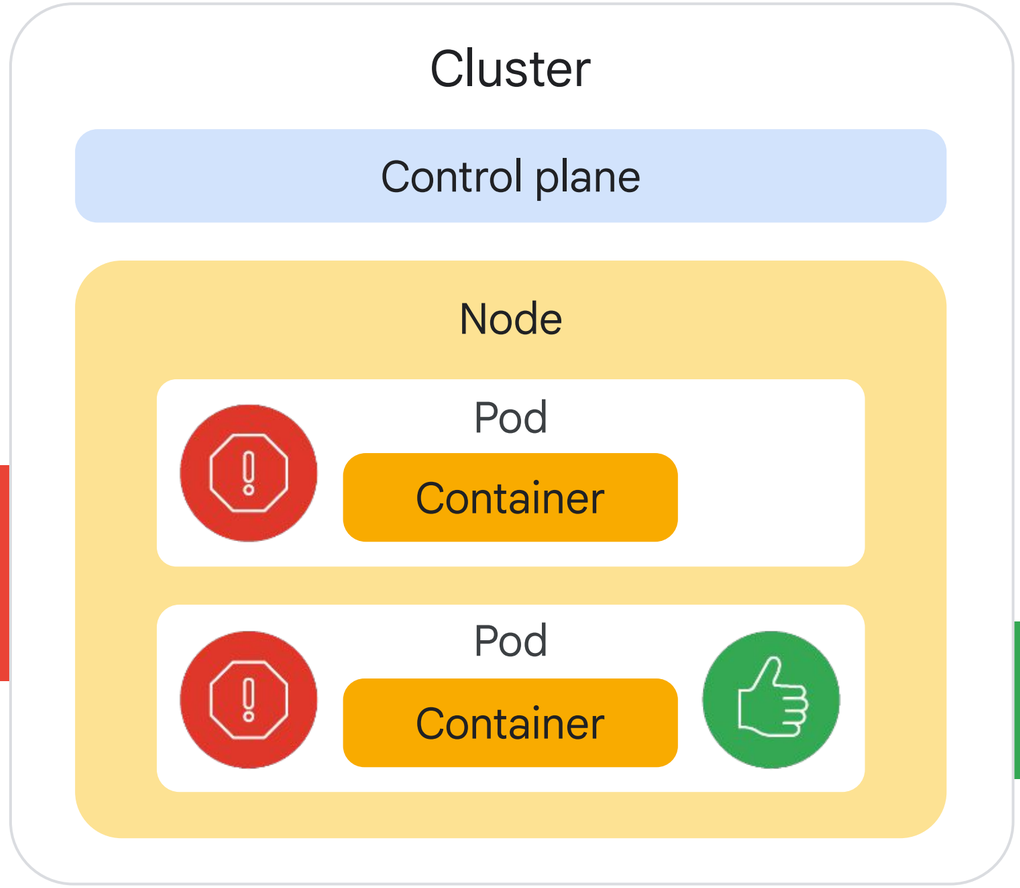
Provides optional information that further clarifies the taint.

Effect

Prevents or discourages Pods from being scheduled on tainted nodes.

A toleration acts like an exception

A taint notifies certain types of Pods that they are not welcome on a node.



A toleration overrides the taint and runs on the node anyway.

A toleration consists of four components

Node

Tainted with:
`Key=Value: NoSchedule`

Pod

```
tolerations:
  - key: “key”
    operator: “Equal”
    value: “value”
    effect: “NoSchedule”
```

Pod

```
tolerations:
  - key: “key”
    operator: “Exists”
    effect: “NoSchedule”
```

Pod

```
tolerations:
  - key: “key”
    operator: “Exists”
    effect: “PreferNoSchedule”
```

A toleration's operator

Pod

```
tolerations:
  - key: “key”
    operator: “Equal”
    value: “value”
    effect: “NoSchedule”
```

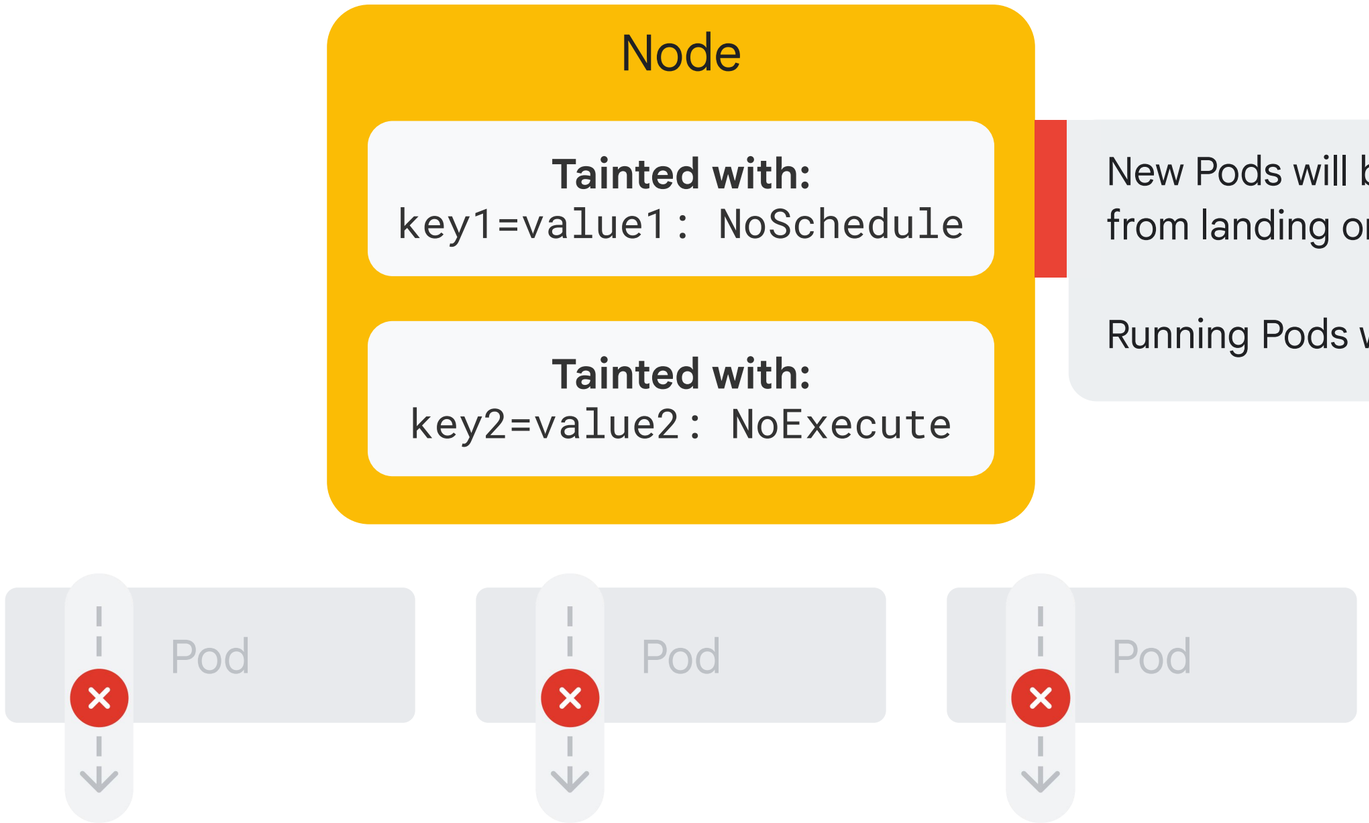
Must match the taint's value exactly to be effective.

Pod

```
tolerations:
  - key: “key”
    operator: “Exists”
    effect: “NoSchedule”
```

Must have the same key and effect as the taint, regardless of value.

What happens when multiple taints are applied to a node



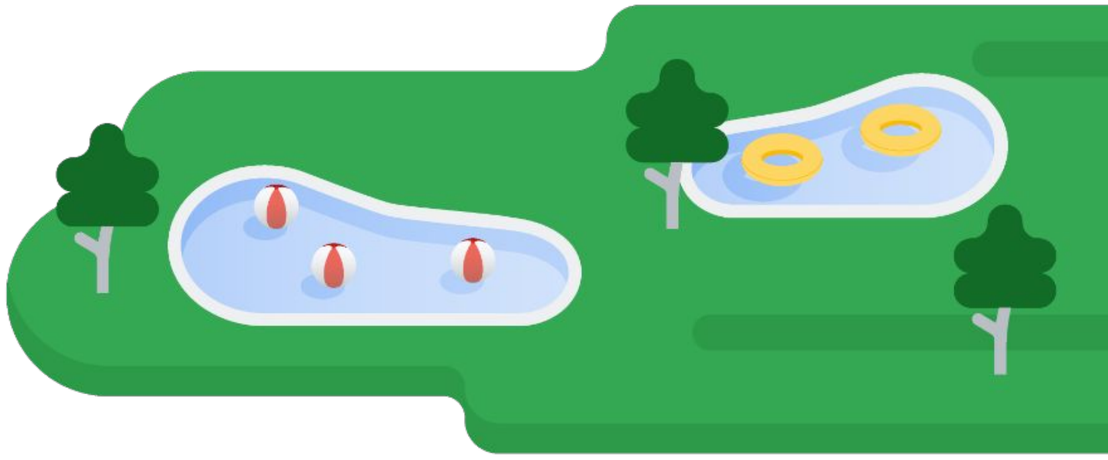
Three effect settings for taints and tolerations

`NoSchedule`	Prevents new Pods from being scheduled on the tainted node, unless the toleration effect is also set to NoSchedule.
`PreferNoSchedule`	Encourages—but does not strictly prohibit—Pod scheduling, which means that the Pod might still be scheduled.
`NoExecute`	Evicts existing Pods and prevents new ones from landing on the tainted node, unless the Pods have a toleration set to NoExecute.

GKE uses node pools to place Pods on the correct hardware

NodeSelector

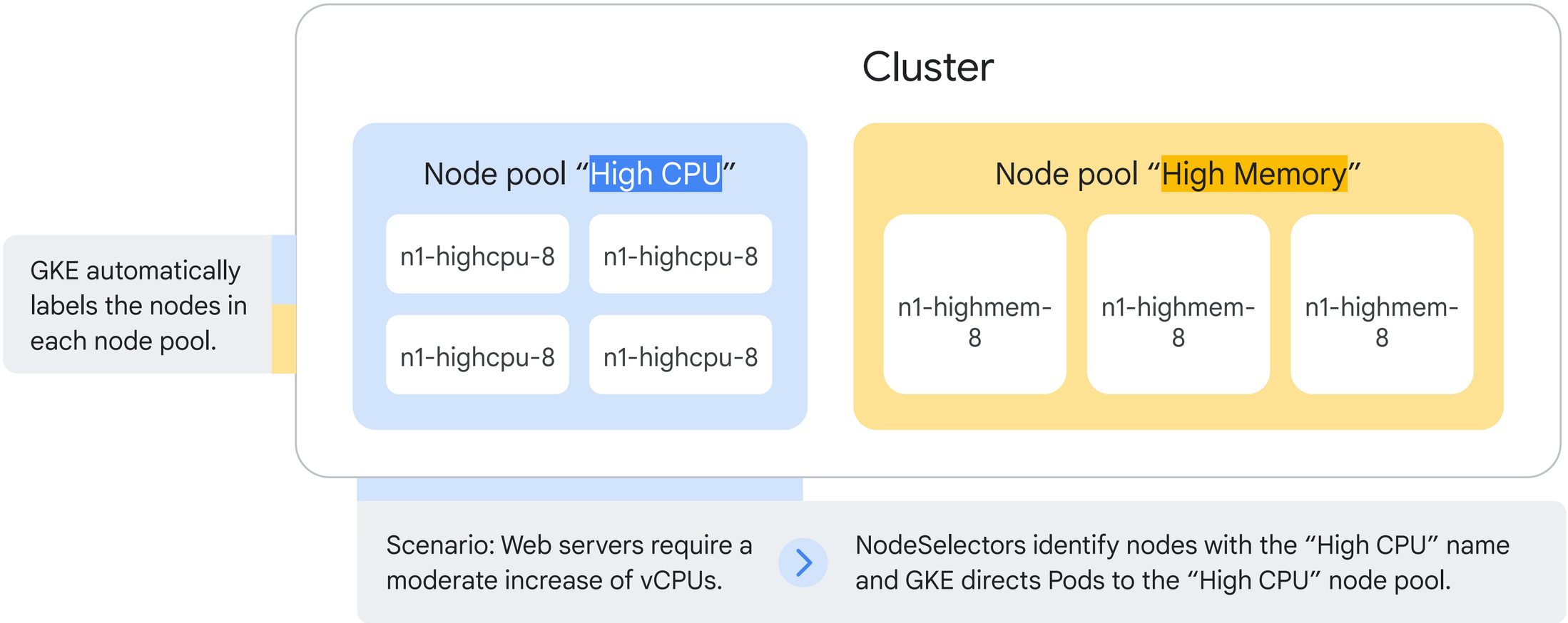
Directs Pods to the correct node pool.



Pods

Node pools

NodeSelectors can direct Pods to nodes



Workloads: Deployments and Jobs

01 Configure, manage, and update Deployments

02 Jobs and CronJobs

03 Scale clusters

04 Control Pod placement with labels and affinity rules

05 Control Pod placement with taints and tolerations

06 Get software into a cluster

Tools to help get software into a cluster

Cloud Build

Serverless tool to build, test, and deploy software across various environments and programming languages.

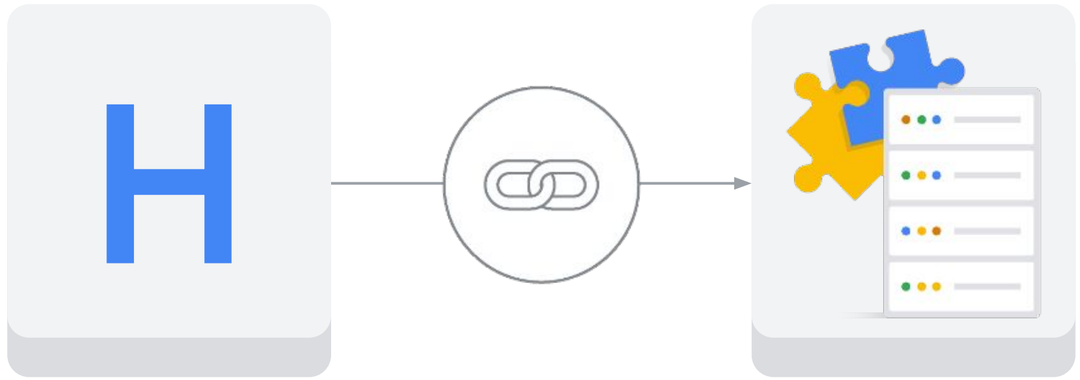
Artifact Registry

Provides a single location to store, manage, and secure build artifacts.

Helm

An open-sourced package manager that simplifies application management.

Helm and the API server



- Install
- Upgrade
- Query
- Delete

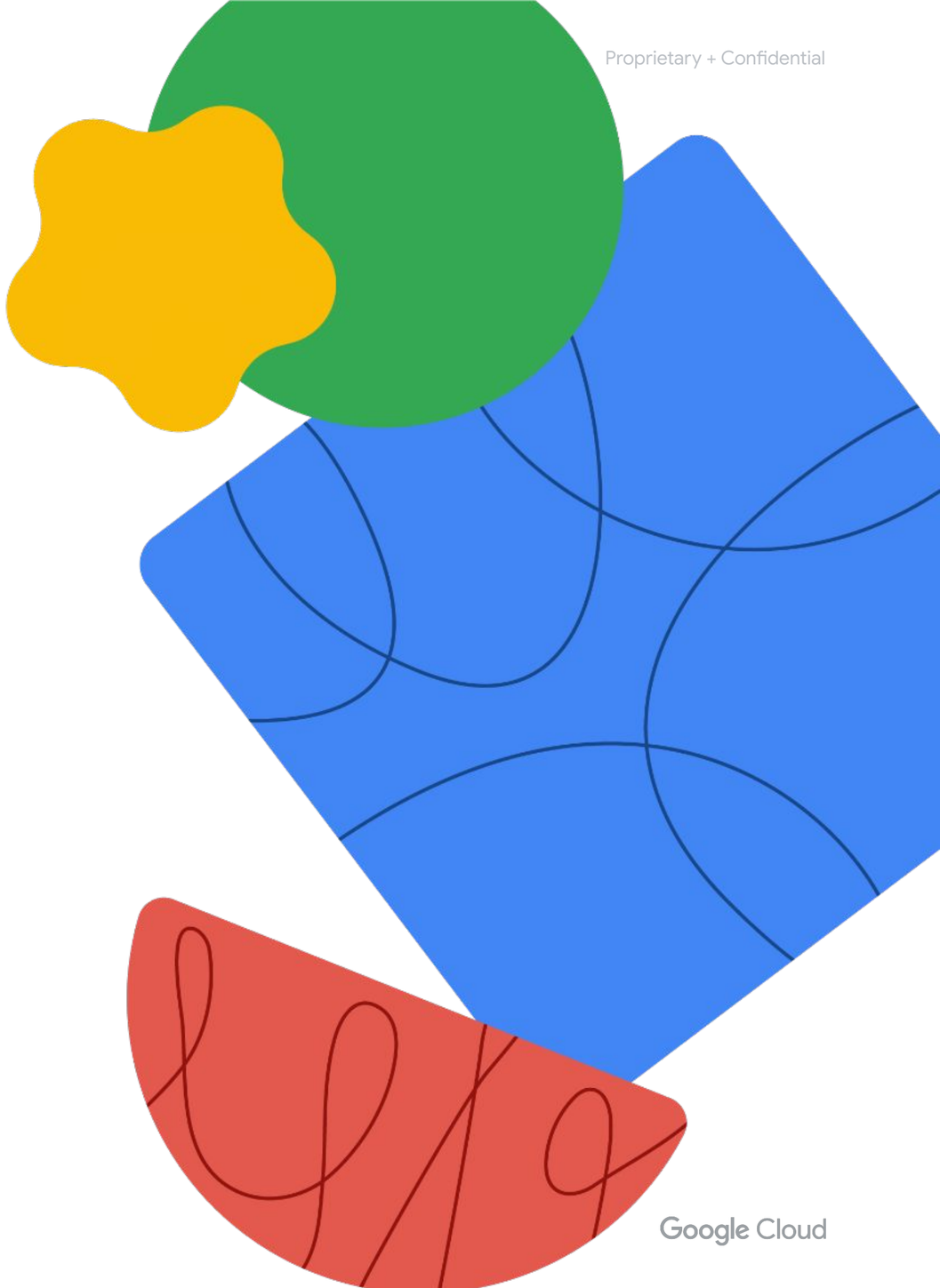
Helm must be managed

Google Cloud Marketplace

- Development stacks
- Solutions
- Services

Quiz questions

Let’s pause for a quick check in.



Quiz | Question 1

Question

After a Deployment has been created and its component Pods are running, which component is responsible for ensuring that a replacement Pod is launched whenever a Pod fails or is evicted?

A. ReplicaSet

B. Deployment

C. StatefulSet

D. DaemonSet

Quiz | Question 1

Answer

After a Deployment has been created and its component Pods are running, which component is responsible for ensuring that a replacement Pod is launched whenever a Pod fails or is evicted?

A. ReplicaSet

B. Deployment

C. StatefulSet

D. DaemonSet

Quiz | Question 2

Question

Which of the following commands will display the desired, current, up-to-date, and available status of all the ReplicaSets within a Deployment?

A. kubectl describe

B. kubectl logs

C. kubectl get

D. Google Cloud Console

Quiz | Question 2

Answer

Which of the following commands will display only the desired, current, up-to-date, and available status of all the ReplicaSets within a Deployment?

A. kubectl describe

B. kubectl logs

C. kubectl get

D. Google Cloud Console

Quiz | Question 3

Question

You are resolving a range of issues with a Deployment and need to make a large number of changes. Which command can you execute to group these changes into a single rollout, thus avoiding pushing out a large number of rollouts?

A. kubectl rollout resume deployment

B. kubectl delete deployment

C. kubectl stop deployment

D. kubectl rollout pause deployment

Quiz | Question 3

Answer

You are resolving a range of issues with a Deployment and need to make a large number of changes. Which command can you execute to group these changes into a single rollout, thus avoiding pushing out a large number of rollouts?

A. kubectl rollout resume deployment

B. kubectl delete deployment

C. kubectl stop deployment

D. kubectl rollout pause deployment

Quiz | Question 4

Question

You are configuring a Job to convert a sample of a large number of video files to a different format. Which parameter should you configure to stop the process once a sufficient quantity is reached?

A. parallelism=4

B. completions=4

C. backofflimit=4

D. replicas=4

Quiz | Question 4

Answer

You are configuring a Job to convert a sample of a large number of video files to a different format. Which parameter should you configure to stop the process once a sufficient quantity is reached?

A. parallelism=4

B. completions=4 

C. backofflimit=4

D. replicas=4

Quiz | Question 5

Question

How do you configure a Kubernetes Job so that Pods are retained after completion?

A. Configure the cascade flag for the Job with a value of false.

B. Configure the backofflimit parameter with a non-zero value.

C. Set an activeDeadlineSeconds value high enough to allow you to access the logs.

D. Set a startingDeadlineSeconds value high enough to allow you to access the logs.

Quiz | Question 5

Answer

How do you configure a Kubernetes Job so that Pods are retained after completion?

A. Configure the cascade flag for the Job with a value of false. 

B. Configure the backofflimit parameter with a non-zero value.

C. Set an activeDeadlineSeconds value high enough to allow you to access the logs.

D. Set a startingDeadlineSeconds value high enough to allow you to access the logs.

Quiz | Question 6

Question

You have autoscaling enabled on your cluster. What conditions are required for the autoscaler to decide to delete a node?

A. If a node is underutilized and there are no Pods currently running on the Node.

B. If the overall cluster is underutilized, the least busy node is deleted.

C. If the overall cluster is underutilized, a randomly selected node is deleted.

D. If a node is underutilized and running Pods can be run on other Nodes.

Quiz | Question 6

Answer

You have autoscaling enabled on your cluster. What conditions are required for the autoscaler to decide to delete a node?

A. If a node is underutilized and there are no Pods currently running on the Node.

B. If the overall cluster is underutilized, the least busy node is deleted.

C. If the overall cluster is underutilized, a randomly selected node is deleted.

D. If a node is underutilized and running Pods can be run on other Nodes.

Quiz | Question 7

Question

Inter-pod affinity rules are specified at the zone level, not at the individual Node level. To apply this override, which additional parameter must be configured in the Pod manifest YAML?

A. zone: topology.kubernetes.io/zone

B. topologyKey: topology.kubernetes.io/zone

C. labels: topology.kubernetes.io/zone

D. matchLabels: topology.kubernetes.io/zone

Quiz | Question 7

Answer

Inter-pod affinity rules are specified at the zone level, not at the individual Node level. To apply this override, which additional parameter must be configured in the Pod manifest YAML?

A. zone: topology.kubernetes.io/zone

B. topologyKey: topology.kubernetes.io/zone 

C. labels: topology.kubernetes.io/zone

D. matchLabels: topology.kubernetes.io/zone