2024 Q1 AI Voice SRE - Readiness Probes

SRE, Site Reliability Engineering, is what you get when you treat operations as if it's a software problem.

Kubernetes Readiness Probes Best Practice

A readiness probe indicates whether applications running in a container are ready to receive traffic. If so, Services in Kubernetes can send traffic to the pod, and if not, the endpoint controller removes the pod from all services.

Kubernetes provides the following types of probes. For all these types, if the container does not implement the probe handler, their result is always Success.

- **Liveness Probe**—indicates if the container is operating. If so, no action is taken. If not, the kubelet kills and restarts the container. Learn more in our guide to Kubernetes liveness probes.
- **Readiness Probe**—indicates whether the application running in the container is ready to accept requests. If so, Services matching the pod are allowed to send traffic to it. If not, the endpoints controller removes the pod from all matching Kubernetes Services.
- **Startup Probe**—indicates whether the application running in the container has started. If so, other probes start functioning. If not, the kubelet kills and restarts the container.

Readiness probes are most useful when an application is temporarily malfunctioning and unable to serve traffic. If the application is running but not fully available, Kubernetes may not be able to scale it up and new deployments could fail. A readiness probe allows Kubernetes to wait until the service is active before sending it traffic.

When you use a readiness probe, keep in mind that Kubernetes will only send traffic to the pod if the probe succeeds.

Readiness Probe Example

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  template:
    metadata:
      labels:
        app: my-test-app
    spec:
      containers:
     –name: my-test-app
        image: nginx:1.14.2
        readinessProbe:
          httpGet:
            path: /ready
            port: 80
          successThreshold: 1
```

Readiness Probe Parameters

PARAMETER	DESCRIPTION	DEFAULT VALUE
`initialDelaySeconds`	Number of seconds between container start and probe start to allow for services to initialize.	0
`periodSeconds`	Frequency of readiness test.	10
`timeoutSeconds`	Timeout for probe responses.	1
`successThreshold`	The number of consecutive success results needed to switch probe status to “Success”.	1
`failureThreshold`	The number of consecutive failed results needed to switch probe status to “Failure”.	3

Common Error Scenarios

Readiness probes are used to verify tasks during a container lifecycle. This means that if the probe's response is interrupted or delayed, service may be interrupted. Keep in mind that if a readiness probe returns Failure status, Kubernetes will remove the pod from all matching service endpoints. Here are two examples of conditions that can cause an application to incorrectly fail the readiness probe.

Delayed Response

In some circumstances, readiness probes may be late to respond—for example, if the application needs to read large amounts of data with low latency or perform heavy computations. Consider this behavior when configuring readiness probes, and always test your application thoroughly before running it in production with a readiness probe.

Cascading Failures

A readiness probe response can be conditional on components that are outside the direct control of the application. For example, you could configure a readiness probe using HTTPGet, in such a way that the application first checks the availability of a cache service or database before responding to the probe. This means that if the database is down or late to respond, the entire application will become unavailable.

This may or may not make sense, depending on your application setup. If the application cannot function at all without the third-party component, maybe this behavior is warranted. If it can continue functioning, for example, by falling back to a local cache, the database or external cache should not be connected to probe responses.

In general, if the pod is technically ready, even if it cannot function perfectly, it should not fail the readiness probe. A good compromise is to implement a “degraded mode,” for example, if there is no access to the database, answer read requests that can be addressed by local cache and return 503 (service unavailable) on write requests. Ensure that downstream services are resilient to a failure in the upstream service.

Troubleshooting Readiness Probes and Other Node Errors

Troubleshooting Kubernetes nodes relies on the ability to quickly contextualize the problem with what’s happening in the rest of the cluster. More often than not, you will be conducting your investigation during fires in production. The major challenge is correlating service-level incidents with other events happening in the underlying infrastructure.

In particular, when nodes go offline due to readiness probe failures, it is necessary to understand what is happening on all nodes involved and get context about other events in the environment that might be significant.

DevOps needs to have the access to view the Kubernetes cluster node status; to pinpoint correlations between service or deployment issues and changes in the underlying node infrastructure. With this view DevOps can rapidly:

- See service-to-node associations
- Correlate service and node health issues
- Gain visibility over node capacity allocations, restrictions, and limitations
- Identify “noisy neighbors” that use up cluster resources.