Kubernetes Stories — Phased End-to-End Workflow

Foundations → Visibility → Automation → Release → Scale → Security → Traffic → Depth → Advanced

1 Workflow Phases (Overview)

Foundations & Dev Experience

- **KBP-01** — Basic service + Helm + ingress (hello world end-to-end).
- **KBP-02** — Developer workflows (namespaces, RBAC, inner loop).
- **KBP-04** — Config & secrets baseline (separate config, non-secret vs secret).

Observability First (make issues visible early)

- **KBP-03** — Monitoring + logging (Prometheus, Grafana, Loki/EFK).

Ship Automatically

- **KBP-05** — CI pipeline to build/test/deploy.
- **KBP-18** — Adopt GitOps for deployments (pipeline hands off to GitOps).

Release Discipline

- **KBP-06** — Versioning, releases, rollout patterns (blue/green, canary).

Efficiency & Stability

- **KBP-08** — Requests/limits, HPA, PDB (right-size + graceful disruptions).
- **KBP-20** — Resilience & performance tests (chaos/load with SLO checks).

Security Baseline & Guardrails

- **KBP-10** — Workload hardening (PSA, seccomp, non-root).
- **KBP-17** — Admission control & authorization (API governance levers).
- **KBP-11** — Policy-as-code (Gatekeeper/Kyverno) to enforce standards.
- **KBP-19** — Holistic security posture (images, SBOMs, scanners, posture views).

Networking & Traffic Control

- **KBP-09** — Networking hardening, Gateway API / service mesh (mTLS, traffic splits).

Scale Out & Multi-Everything

- **KBP-07** — Multi-region staging & controlled rollout (regional values/canaries).
- **KBP-12** — Multi-cluster management with GitOps (fleet-level practices).

App/Platform Depth

- **KBP-16** — Stateful services (storage, backups, operators).
- **KBP-13** — External integrations (safe service-to-service, egress controls).
- **KBP-15** — Higher-level app/platform patterns (golden paths, templates).

Advanced & Extensibility

- **KBP-21** — Build a simple operator (Kubebuilder) to encode ops runbooks.
- **KBP-14** — ML inference on K8s (only after observability, security, scaling are in).

Wrap-Up & Forward Plan

- **KBP-22** — Document conclusions & next-90-day roadmap (what to double-down on).

2 Sequenced Story Index by Phase

Foundations & Dev Experience

1. **K8S-01** — Get a Local Cluster Running
2. **K8S-02** — Provision Clusters (kubeadm + managed)
3. **K8S-03** — Master `kubectl` Fundamentals
4. **K8S-04** — Deploy Core Workloads
5. **K8S-06** — Package with Helm & Friends
6. **K8S-07** — Govern with Namespaces/Quotas
7. **K8S-08** — Persist & Configure Safely

Observability First

1. **K8S-11** — Observe Health & Behavior (metrics/logs/traces, dashboards/alerts)

Ship Automatically

- *Add your CI & GitOps stories here if present (e.g., KBP-05, KBP-18).*

Release Discipline

1. **K8S-05** — Expose Applications Reliably (Ingress/Gateway, TLS, rollout patterns)

Efficiency & Stability

1. **K8S-09** — Autoscale Workloads (HPA, right-size requests/limits)
2. **K8S-12** — Diagnose & Repair Fast (troubleshooting runbooks, SLO-driven fixes)

Security Baseline & Guardrails

1. **K8S-10** — Enforce Least Privilege (RBAC, securityContext, PSA/Pod Security)

Networking & Traffic Control

1. **K8S-05** — Expose Applications Reliably (applies here for traffic policy)
2. **K8S-13** — Introduce Mesh Traffic Control (mTLS, retries, timeouts, splits)

Scale Out & Multi-Everything

- *Add multi-region/multi-cluster GitOps stories here if present (e.g., KBP-07, KBP-12).*

App/Platform Depth

1. **K8S-14** — Scale-to-Zero with Knative (eventing/serving)
2. **K8S-15** — Build/Extend the Platform (golden paths, templates)

Advanced & Extensibility

- *Add operator/ML stories here if present (e.g., KBP-21, KBP-14).*

Wrap-Up & Forward Plan

- *Retrospective, platform scorecard, and next-90-day roadmap.*

Kubernetes Stories — End-to-End Workflow

Sequenced, dependency-aware path from local lab to platform operation

Workflow at a Glance

1. K8S-01 — Get a Local Cluster Running
2. K8S-02 — Provision Clusters (kubeadm + managed)
3. K8S-03 — Master kubectl Fundamentals
4. K8S-04 — Deploy Core Workloads
5. K8S-05 — Expose Applications Reliably
6. K8S-06 — Package with Helm & Friends
7. K8S-08 — Persist & Configure Safely
8. K8S-07 — Govern with Namespaces/Quotas
9. K8S-10 — Enforce Least Privilege
10. K8S-11 — Observe Health & Behavior
11. K8S-09 — Autoscale Workloads
12. K8S-12 — Diagnose & Repair Fast

3 Getting Started with Kubernetes

K8S-01 — Get a Local Cluster Running

Epic / Feature	Kubernetes Basics
Business Value	Establish a reproducible local lab to safely experiment and learn
Priority / Estimate	Priority: Must SP: 3
Persona	developer
Dependencies	Docker, `kubectl`, `kind` or `minikube`
Assumptions / Risks	Resource constraints on laptop; network/proxy issues

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *developer*, I want to get a Local Cluster Running so that *Establish a reproducible local lab to safely experiment and learn.*

Acceptance Criteria (BDD)

```
    Scenario: Install \texttt{kubectl} and \texttt{kind} or \texttt{miniku
  Given Docker, \texttt{kubectl}, \texttt{kind} or \texttt{minikube}
  When Install \texttt{kubectl} and \texttt{kind} or \texttt{minikube}; verify
    \texttt{kubectl version} and context
  Then expected outcome is observable in logs/CLI/UI

Scenario: Create a cluster; enable metrics-server (minikube addon or Y
  Given Docker, \texttt{kubectl}, \texttt{kind} or \texttt{minikube}
  When Create a cluster; enable metrics-server (minikube addon or YAML)
  Then expected outcome is observable in logs/CLI/UI

Scenario: Deploy a sample Deployment + Service; confirm Pod readiness
  Given Docker, \texttt{kubectl}, \texttt{kind} or \texttt{minikube}
  When Deploy a sample Deployment + Service; confirm Pod readiness and Service
    reachability
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Install `kubectl` and `kind` or `minikube`; verify `kubectl version` and context.
- ☐ Create a cluster; enable metrics-server (minikube addon or YAML).
- ☐ Deploy a sample Deployment + Service; confirm Pod readiness and Service reachability.
- ☐ Capture a cheatsheet of 20 `kubectl` commands in `/labs/ch01/README.md`.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

4 Creating a Kubernetes Cluster

K8S-02 — Provision Clusters (kubeadm + managed)

Epic / Feature	Cluster Provisioning
Business Value	Understand DIY vs. managed tradeoffs and create a repeatable runbook
Priority / Estimate	Priority: Must SP: 5
Persona	platform engineer
Dependencies	Linux VMs, cloud account (GKE/EKS/AKS), CNI
Assumptions / Risks	Quota/permissions in cloud; VM CPU/mem limits

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *platform engineer*, I want to provision Clusters so that *Understand DIY vs. managed tradeoffs and create a repeatable runbook.*

Acceptance Criteria (BDD)

```
    Scenario: Bootstrap a single-control-plane cluster via \texttt{kubeadm
  Given Linux VMs, cloud account (GKE/EKS/AKS), CNI
  When Bootstrap a single-control-plane cluster via \texttt{kubeadm}; install a CNI
  Then expected outcome is observable in logs/CLI/UI

Scenario: Join a worker; validate node readiness; label/taint as neede
  Given Linux VMs, cloud account (GKE/EKS/AKS), CNI
  When Join a worker; validate node readiness; label/taint as needed
  Then expected outcome is observable in logs/CLI/UI

Scenario: Create one managed cluster (pick a cloud); install metrics-s
  Given Linux VMs, cloud account (GKE/EKS/AKS), CNI
  When Create one managed cluster (pick a cloud); install metrics-server \& Dashboard
    (protected)
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Bootstrap a single-control-plane cluster via `kubeadm`; install a CNI.
- ☐ Join a worker; validate node readiness; label/taint as needed.
- ☐ Create one managed cluster (pick a cloud); install metrics-server & Dashboard (protected).
- ☐ Write a `create/destroy` runbook for both environments.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

5 Learning to Use the Kubernetes Client

K8S-03 — Master kubectl Fundamentals

Epic / Feature	Developer Experience
Business Value	Reduce MTTR and increase flow via fluent CLI usage
Priority / Estimate	Priority: Must SP: 2
Persona	developer
Dependencies	Context/namespace helpers
Assumptions / Risks	Risk of destructive commands; use `-dry-run=client`

Non-Functional

Performance   Security   Reliability   Accessibility   Privacy   i18n

Story: As a *developer*, I want to master kubectl Fundamentals so that *Reduce MTTR and increase flow via fluent CLI usage.*

Acceptance Criteria (BDD)

```
    Scenario: Practice \texttt{get}, \texttt{describe}, \texttt{logs}, \te
  Given Context/namespace helpers
  When Practice \texttt{get}, \texttt{describe}, \texttt{logs}, \texttt{exec},
    \texttt{delete --cascade}
  Then expected outcome is observable in logs/CLI/UI

Scenario: Use \texttt{kubectl explain} and JSONPath queries; export YA
  Given Context/namespace helpers
  When Use \texttt{kubectl explain} and JSONPath queries; export YAML via \texttt{-o
    yaml}
  Then expected outcome is observable in logs/CLI/UI

Scenario: Create namespace shortcuts (\texttt{kubens}) and context swi
  Given Context/namespace helpers
  When Create namespace shortcuts (\texttt{kubens}) and context switches
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Practice `get`, `describe`, `logs`, `exec`, `delete -cascade`.
- ☐ Use `kubectl explain` and JSONPath queries; export YAML via `-o yaml`.
- ☐ Create namespace shortcuts (`kubens`) and context switches.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

6 Creating and Modifying Fundamental Workloads

K8S-04 — Deploy Core Workloads

Epic / Feature	Workload Primitives
Business Value	Safely roll out, pause, and roll back application changes
Priority / Estimate	Priority: Must SP: 3
Persona	app developer
Dependencies	Container image registry
Assumptions / Risks	Image pull limits; tag discipline

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *app developer*, I want to deploy Core Workloads so that *Safely roll out, pause, and roll back application changes.*

Acceptance Criteria (BDD)

```
    Scenario: Create Pod, Deployment (with rolling update), Job, CronJob,
  Given Container image registry
  When Create Pod, Deployment (with rolling update), Job, CronJob, DaemonSet examples
  Then expected outcome is observable in logs/CLI/UI

Scenario: Trigger a rollout; verify \texttt{rollout status/history}; p
  Given Container image registry
  When Trigger a rollout; verify \texttt{rollout status/history}; perform rollback
  Then expected outcome is observable in logs/CLI/UI

Scenario: Add \texttt{readiness/liveness} probes to one Deployment
  Given Container image registry
  When Add \texttt{readiness/liveness} probes to one Deployment
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Create Pod, Deployment (with rolling update), Job, CronJob, DaemonSet examples.
- ☐ Trigger a rollout; verify `rollout status/history`; perform rollback.
- ☐ Add `readiness/liveness` probes to one Deployment.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

7 Working with Services

K8S-05 — Expose Applications Reliably

Epic / Feature	Networking & Discovery
Business Value	Provide stable service discovery and ingress to users
Priority / Estimate	Priority: Must SP: 3
Persona	application SRE
Dependencies	CoreDNS, Ingress controller
Assumptions / Risks	Ingress misconfig; path conflicts

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *application SRE*, I want to expose Applications Reliably so that *Provide stable service discovery and ingress to users.*

Acceptance Criteria (BDD)

```
    Scenario: Create ClusterIP/NodePort/LoadBalancer Services and compare
  Given CoreDNS, Ingress controller
  When Create ClusterIP/NodePort/LoadBalancer Services and compare
  Then expected outcome is observable in logs/CLI/UI

Scenario: Install NGINX Ingress; route \texttt{/} to an app; verify fr
  Given CoreDNS, Ingress controller
  When Install NGINX Ingress; route \texttt{/} to an app; verify from host
  Then expected outcome is observable in logs/CLI/UI

Scenario: Validate DNS inside Pods using \texttt{nslookup} or \texttt{
  Given CoreDNS, Ingress controller
  When Validate DNS inside Pods using \texttt{nslookup} or \texttt{dig}
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Create ClusterIP/NodePort/LoadBalancer Services and compare.
- ☐ Install NGINX Ingress; route `/` to an app; verify from host.
- ☐ Validate DNS inside Pods using `nslookup` or `dig`.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

8 Managing Application Manifests

K8S-06 — Package with Helm & Friends

Epic / Feature	Deployment Packaging
Business Value	Enable repeatable, parameterized deployments across envs
Priority / Estimate	Priority: Must SP: 5
Persona	platform engineer
Dependencies	Helm, optional: Kompose/Carvel
Assumptions / Risks	Values drift; document overrides

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *platform engineer*, I want to package with Helm & Friends so that *Enable repeatable, parameterized deployments across envs.*

Acceptance Criteria (BDD)

```
    Scenario: Install Helm; deploy a public chart with custom \texttt{valu
  Given Helm, optional: Kompose/Carvel
  When Install Helm; deploy a public chart with custom \texttt{values.yaml}
  Then expected outcome is observable in logs/CLI/UI

Scenario: Convert a simple docker-compose app using \texttt{kompose};
  Given Helm, optional: Kompose/Carvel
  When Convert a simple docker-compose app using \texttt{kompose}; compare output
  Then expected outcome is observable in logs/CLI/UI

Scenario: Author a tiny Helm chart for your sample app; include Notes
  Given Helm, optional: Kompose/Carvel
  When Author a tiny Helm chart for your sample app; include Notes and README
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Install Helm; deploy a public chart with custom `values.yaml`.
- ☐ Convert a simple docker-compose app using `kompose`; compare output.
- ☐ Author a tiny Helm chart for your sample app; include Notes and README.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

9 Volumes and Configuration Data

K8S-08 — Persist & Configure Safely

Epic / Feature	State & Config
Business Value	Separate config/secrets from code and preserve state across restarts
Priority / Estimate	Priority: Must SP: 5
Persona	app developer
Dependencies	ConfigMap, Secret, PV/PVC
Assumptions / Risks	Secret sprawl; adopt rotation practices

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *app developer*, I want to persist & Configure Safely so that *Separate config/secrets from code and preserve state across restarts.*

Acceptance Criteria (BDD)

```
    Scenario: Mount ConfigMap values; inject a Secret (env or volume)
  Given ConfigMap, Secret, PV/PVC
  When Mount ConfigMap values; inject a Secret (env or volume)
  Then expected outcome is observable in logs/CLI/UI

Scenario: Create a PVC; verify data survives Pod restarts
  Given ConfigMap, Secret, PV/PVC
  When Create a PVC; verify data survives Pod restarts
  Then expected outcome is observable in logs/CLI/UI

Scenario: Document secret handling (at-rest encryption, \texttt{.docke
  Given ConfigMap, Secret, PV/PVC
  When Document secret handling (at-rest encryption, \texttt{.dockerconfigjson},
    \texttt{imagePullSecrets})
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Mount ConfigMap values; inject a Secret (env or volume).
- ☐ Create a PVC; verify data survives Pod restarts.
- ☐ Document secret handling (at-rest encryption, `.dockerconfigjson`, `imagePullSecrets`).

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

10 Exploring the Kubernetes API and Key Metadata

K8S-07 — Govern with Namespaces/Quotas

Epic / Feature	API & Metadata
Business Value	Constrain resource usage and organize multi-team tenancy
Priority / Estimate	Priority: Should SP: 3
Persona	platform engineer
Dependencies	ResourceQuota, LimitRange
Assumptions / Risks	Overly strict quotas can block deploys

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *platform engineer*, I want to govern with Namespaces/Quotas so that *Constrain resource usage and organize multi-team tenancy.*

Acceptance Criteria (BDD)

```
    Scenario: List resources via \texttt{kubectl api-resources}; inspect v
  Given ResourceQuota, LimitRange
  When List resources via \texttt{kubectl api-resources}; inspect versions
  Then expected outcome is observable in logs/CLI/UI

Scenario: Create Namespaces with \texttt{ResourceQuota} and \texttt{Li
  Given ResourceQuota, LimitRange
  When Create Namespaces with \texttt{ResourceQuota} and \texttt{LimitRange}
  Then expected outcome is observable in logs/CLI/UI

Scenario: Label/annotate resources; query with selectors
  Given ResourceQuota, LimitRange
  When Label/annotate resources; query with selectors
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ List resources via `kubectl api-resources`; inspect versions.
- ☐ Create Namespaces with `ResourceQuota` and `LimitRange`.
- ☐ Label/annotate resources; query with selectors.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

11 Security

K8S-10 — Enforce Least Privilege

Epic / Feature	Platform Security
Business Value	Reduce blast radius through RBAC and Pod hardening
Priority / Estimate	Priority: Must SP: 5
Persona	security champion
Dependencies	ServiceAccount, Role/Binding, Pod Security Standards
Assumptions / Risks	Permissions confusion; validate with `can-i`

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *security champion*, I want to enforce Least Privilege so that *Reduce blast radius through RBAC and Pod hardening*.

Acceptance Criteria (BDD)

```
    Scenario: Create a dedicated ServiceAccount for an app; bind minimal R
  Given ServiceAccount, Role/Binding, Pod Security Standards
  When Create a dedicated ServiceAccount for an app; bind minimal Role
  Then expected outcome is observable in logs/CLI/UI

Scenario: Add \texttt{securityContext}: \texttt{runAsNonRoot}, \texttt
  Given ServiceAccount, Role/Binding, Pod Security Standards
  When Add \texttt{securityContext}: \texttt{runAsNonRoot},
    \texttt{readOnlyRootFilesystem}, drop caps
  Then expected outcome is observable in logs/CLI/UI

Scenario: Apply Pod Security admission (baseline/restricted) at namesp
  Given ServiceAccount, Role/Binding, Pod Security Standards
  When Apply Pod Security admission (baseline/restricted) at namespace level
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Create a dedicated ServiceAccount for an app; bind minimal Role.
- ☐ Add `securityContext`: `runAsNonRoot`, `readOnlyRootFilesystem`, drop caps.
- ☐ Apply Pod Security admission (baseline/restricted) at namespace level.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

12 Monitoring and Logging

K8S-11 — Observe Health & Behavior

Epic / Feature	Observability
Business Value	Shorten detection time with probes, metrics, and dashboards
Priority / Estimate	Priority: Must SP: 5
Persona	SRE
Dependencies	Probes, kube-prometheus-stack/Grafana
Assumptions / Risks	Dashboard noise; focus on SLI panels

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *SRE*, I want to observe Health & Behavior so that *Shorten detection time with probes, metrics, and dashboards.*

Acceptance Criteria (BDD)

```
    Scenario: Add liveness/readiness/startup probes; induce failures to se
  Given Probes, kube-prometheus-stack/Grafana
  When Add liveness/readiness/startup probes; induce failures to see effects
  Then expected outcome is observable in logs/CLI/UI

Scenario: Deploy Prometheus+Grafana on local cluster; import a simple
  Given Probes, kube-prometheus-stack/Grafana
  When Deploy Prometheus+Grafana on local cluster; import a simple dashboard
  Then expected outcome is observable in logs/CLI/UI

Scenario: Collect and link logs for a failing Pod in \texttt{/labs/ch1
  Given Probes, kube-prometheus-stack/Grafana
  When Collect and link logs for a failing Pod in \texttt{/labs/ch11/README.md}
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Add liveness/readiness/startup probes; induce failures to see effects.
- ☐ Deploy Prometheus+Grafana on local cluster; import a simple dashboard.
- ☐ Collect and link logs for a failing Pod in `/labs/ch11/README.md`.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

13 Scaling

K8S-09 — Autoscale Workloads

Epic / Feature	Capacity & Efficiency
Business Value	Match resources to demand and control costs
Priority / Estimate	Priority: Should SP: 3
Persona	SRE
Dependencies	Metrics Server; optional Cluster Autoscaler
Assumptions / Risks	HPA signals noisy; smooth with requests/limits

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *SRE*, I want to autoscale Workloads so that *Match resources to demand and control costs.*

Acceptance Criteria (BDD)

```
    Scenario: Set CPU/memory requests and limits; run a small load test
  Given Metrics Server; optional Cluster Autoscaler
  When Set CPU/memory requests and limits; run a small load test
  Then expected outcome is observable in logs/CLI/UI

Scenario: Configure HPA; observe scale-out/back with \texttt{kubectl t
  Given Metrics Server; optional Cluster Autoscaler
  When Configure HPA; observe scale-out/back with \texttt{kubectl top}
  Then expected outcome is observable in logs/CLI/UI

Scenario: (Cloud) Enable Cluster Autoscaler; capture event timeline
  Given Metrics Server; optional Cluster Autoscaler
  When (Cloud) Enable Cluster Autoscaler; capture event timeline
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Set CPU/memory requests and limits; run a small load test.
- ☐ Configure HPA; observe scale-out/back with `kubectl top`.
- ☐ (Cloud) Enable Cluster Autoscaler; capture event timeline.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

14 Maintenance and Troubleshooting

K8S-12 — Diagnose & Repair Fast

Epic / Feature	Ops Readiness
Business Value	Reduce MTTR with systematic debugging and safe maintenance
Priority / Estimate	Priority: Must SP: 5
Persona	SRE
Dependencies	kubectl debug / drain
Assumptions / Risks	Node drains can disrupt; use PodDisruptionBudgets

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *SRE*, I want to diagnose & Repair Fast so that *Reduce MTTR with systematic debugging and safe maintenance.*

Acceptance Criteria (BDD)

```
    Scenario: Reproduce common failures: CrashLoopBackOff, Pending PVC, Im
  Given kubectl debug / drain
  When Reproduce common failures: CrashLoopBackOff, Pending PVC, ImagePullBackOff
  Then expected outcome is observable in logs/CLI/UI

Scenario: Use \texttt{kubectl debug} or ephemeral containers to inspec
  Given kubectl debug / drain
  When Use \texttt{kubectl debug} or ephemeral containers to inspect
  Then expected outcome is observable in logs/CLI/UI

Scenario: Practice \texttt{cordon/drain/uncordon}; snapshot cluster st
  Given kubectl debug / drain
  When Practice \texttt{cordon/drain/uncordon}; snapshot cluster state
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Reproduce common failures: CrashLoopBackOff, Pending PVC, ImagePullBackOff.
- ☐ Use `kubectl debug` or ephemeral containers to inspect.
- ☐ Practice `cordon/drain/uncordon`; snapshot cluster state.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

15 Service Meshes

K8S-13 — Introduce Mesh Traffic Control

Epic / Feature	Service Mesh
Business Value	Gain mTLS, traffic shaping, and better service insights
Priority / Estimate	Priority: Could SP: 5
Persona	platform engineer
Dependencies	Istio or Linkerd
Assumptions / Risks	Sidecar overhead; start small

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *platform engineer*, I want to introduce Mesh Traffic Control so that *Gain mTLS, traffic shaping, and better service insights.*

Acceptance Criteria (BDD)

```
    Scenario: Install Istio or Linkerd; enable automatic sidecar injection
  Given Istio or Linkerd
  When Install Istio or Linkerd; enable automatic sidecar injection
  Then expected outcome is observable in logs/CLI/UI

Scenario: Implement a canary (90/10 \textrightarrow{} 50/50 \textright
  Given Istio or Linkerd
  When Implement a canary (90/10 \textrightarrow{} 50/50 \textrightarrow{} 0/100);
    confirm mTLS
  Then expected outcome is observable in logs/CLI/UI

Scenario: Capture latency/error-rate before/after in notes
  Given Istio or Linkerd
  When Capture latency/error-rate before/after in notes
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Install Istio or Linkerd; enable automatic sidecar injection.
- ☐ Implement a canary (90/10 → 50/50 → 0/100); confirm mTLS.
- ☐ Capture latency/error-rate before/after in notes.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

16 Serverless and Event-Driven Applications

K8S-14 — Scale-to-Zero with Knative

Epic / Feature	Serverless & Events
Business Value	Lower infra costs and simplify event plumbing
Priority / Estimate	Priority: Could SP: 5
Persona	app developer
Dependencies	Knative Serving/Eventing; optional TriggerMesh
Assumptions / Risks	Cold starts; set expectations

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *app developer*, I want to scale-to-Zero with Knative so that *Lower infra costs and simplify event plumbing.*

Acceptance Criteria (BDD)

```
    Scenario: Install Knative; deploy a Knative Service; validate scale-to
  Given Knative Serving/Eventing; optional TriggerMesh
  When Install Knative; deploy a Knative Service; validate scale-to-zero
  Then expected outcome is observable in logs/CLI/UI

Scenario: Wire an event source \textrightarrow{} broker \textrightarro
  Given Knative Serving/Eventing; optional TriggerMesh
  When Wire an event source \textrightarrow{} broker \textrightarrow{} trigger
    \textrightarrow{} consumer
  Then expected outcome is observable in logs/CLI/UI

Scenario: Diagram the event flow and save with manifests
  Given Knative Serving/Eventing; optional TriggerMesh
  When Diagram the event flow and save with manifests
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Install Knative; deploy a Knative Service; validate scale-to-zero.
- ☐ Wire an event source → broker → trigger → consumer.
- ☐ Diagram the event flow and save with manifests.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.

17 Extending Kubernetes

K8S-15 — Build/Extend the Platform

Epic / Feature	Platform Extension
Business Value	Tailor Kubernetes via clients, builds, and CRDs
Priority / Estimate	Priority: Should SP: 5
Persona	platform engineer
Dependencies	Go/Python client; CRD scaffolding
Assumptions / Risks	API changes; pin versions

Non-Functional

Performance Security Reliability Accessibility Privacy i18n **Story:** As a *platform engineer*, I want to build/Extend the Platform so that *Tailor Kubernetes via clients, builds, and CRDs.*

Acceptance Criteria (BDD)

```
    Scenario: Compile \texttt{kubectl} locally or build a component from s
  Given Go/Python client; CRD scaffolding
  When Compile \texttt{kubectl} locally or build a component from source
  Then expected outcome is observable in logs/CLI/UI

Scenario: Write a short Python client that watches Pod events
  Given Go/Python client; CRD scaffolding
  When Write a short Python client that watches Pod events
  Then expected outcome is observable in logs/CLI/UI

Scenario: Define a simple CRD; create/list instances via \texttt{kubec
  Given Go/Python client; CRD scaffolding
  When Define a simple CRD; create/list instances via \texttt{kubectl}
  Then expected outcome is observable in logs/CLI/UI
```

Tasks

- ☐ Compile `kubectl` locally or build a component from source.
- ☐ Write a short Python client that watches Pod events.
- ☐ Define a simple CRD; create/list instances via `kubectl`.

Definition of Ready: Persona clear; AC drafted; Dependencies known; Estimate set. • **Definition of Done:** All ACs pass; Tests green; Security/a11y checks; Docs updated; Deployed/flagged.