Untitled

Let's step back and look at the roles of each component:

🏗️ Roles in the Stack

Port.io (Internal Developer Platform / IDP)

- **The Face for Developers**:
  - Provides a **self-service catalog** for developers (e.g., "Deploy App", "Rollback App", "Request Env").
  - Abstracts away *how* things are deployed (ArgoCD, Orkes, Vault).
  - Exposes actions in a nice UI or API developers can safely use.
- **Governance & Guardrails**:
  - Defines **which actions are available** (Sync, Rollback, Deploy Revision).
  - Enforces RBAC/approval policies for who can trigger what.
  - Logs/metadata for compliance and audit.
- **Integration Hub**:
  - Pulls data from ArgoCD, Git, CI/CD pipelines, clusters, etc.
  - Shows developers the state of their apps in one place.

👉 Think of Port as the **Developer Portal & Control Plane**.

Orkes Conductor

- **The Workflow Engine**:
  - Actually executes the **multi-step workflows** behind a Port action.
  - Talks to ArgoCD, Vault, Slack, Jira, Terraform, etc.
  - Handles retries, compensation, approvals, and async flows.

👉 Think of Orkes as the **Brains / Orchestrator**.

ArgoCD

- **The GitOps Engine**:
  - Deploys manifests from Git to Kubernetes clusters.
  - Syncs, rolls back, diffs, etc.

👉 Think of ArgoCD as the **Deployment Engine**.

Vault (via ESO)

- **The Secrets Engine**:
  - Stores ArgoCD tokens, repo creds, cluster creds.
  - ESO keeps K8s secrets in sync securely.

👉 Vault = **Source of Truth for Secrets**.

⚖️ Without Port.io

You'd still have Orkes → ArgoCD → Vault. It would work, but:

- Developers would have to **interact with Orkes directly** (ugly UX).
- You'd lack a **central self-service portal** where all infra actions live.
- No **governance/UI/metadata layer** — just workflows in Orkes.

✅ With Port.io

- Developers go to Port UI and click **"Sync my App"**.
- Port authenticates the user, checks RBAC, logs the action.
- Port calls the correct Orkes workflow.
- Orkes runs the actual orchestration (secrets, ArgoCD calls, retries).
- Port surfaces back the status/result in the developer's context.

🧩 So the Use of Port in This Setup Is:

2394. **Self-Service Developer UX** (simple UI instead of CLI/API calls).
2395. **Governance Layer** (RBAC, approvals, guardrails).
2396. **Audit & Metadata** (who triggered what, when, on which app).
2397. **Integration Point** (unified catalog that combines Git repos, ArgoCD apps, pipelines, etc).
2398. **Abstraction** (developers don't care if it's Orkes or Node.js behind the scenes — they just "click a button").

💡 A good mental model:

- **Port = App Store for developers (UI/UX, policies, catalog).**
- **Orkes = App runtime that actually executes the workflows.**
- **ArgoCD = Specialized subsystem (GitOps deploys).**
- **Vault = Secrets provider.**

summary in plain terms:

📝 Summary: Port + Orkes + ArgoCD + Vault

- **Vault** → Stores all sensitive credentials (e.g., ArgoCD tokens, repo creds).
- **ArgoCD** → Actually deploys, syncs, or rolls back applications in Kubernetes (GitOps engine).
- **Orkes Conductor** → Runs the workflows that *use Vault secrets* and *call ArgoCD APIs*. Handles retries, approvals, error handling.
- **Port.io** → Provides the **developer-facing self-service portal**. Developers click buttons like *"Sync App"* or *"Rollback"*. Port enforces RBAC, governance, and logs.

🚦 Flow Example (Sync App)

638. Developer clicks **Sync App** in Port.
639. Port checks RBAC, logs the action, and triggers the corresponding **Orkes workflow**.
640. Orkes workflow fetches the **ArgoCD token from Vault** and calls ArgoCD API to sync.
641. Orkes handles retries, errors, approvals if needed.
642. Port surfaces back the status/result to the developer in the UI.

🎯 Why This Setup?

- **Port** = Self-service UX + governance.
- **Orkes** = Workflow engine/orchestration logic.
- **ArgoCD** = GitOps deploy engine.
- **Vault** = Secure secret management.

👉 In short: **Port gives developers a safe, governed way to request deployments; Orkes does the heavy lifting; ArgoCD applies the changes; Vault keeps secrets safe.**

Developer clicks "Sync" in Port.io

Port.io triggers Jenkins job

Jenkins pipeline reads ArgoCD token from Vault

Jenkins job calls ArgoCD API (sync/rollback/deploy)

ArgoCD updates application

▼

Jenkins captures response and updates Port.io status

Pros & Cons

Pros of Jenkins:

- Many teams already know Jenkins pipelines.
- Integrates easily with existing CI/CD.
- Flexible: can do build, test, deployment in the same job.

Cons of Jenkins vs Orkes:

- Requires more scripting to mimic Orkes-style workflow orchestration.
- Port.io integration may need custom HTTP steps or a plugin.
- Async status reporting is more manual.

✅ Key Observations

Feature	Orkes	Jenkins
Async retries	✅ Native	⚠ Manual scripting
Fan-out / parallel	✅ Easy	⚠ Limited, parallel stages only
Port.io integration	✅ Simple webhook → workflow	⚠ Needs custom HTTP steps
Secret rotation	✅ CSI-mounted + dynamic	⚠ Plugin or script-based
Maintainability	✅ JSON workflows	⚠ Pipeline scripts can drift

💡 **Takeaway:**

- **Orkes:** Best for complex, secure, and maintainable workflow orchestration.
- **Jenkins:** Fine because the team already has a heavy Jenkins CI/CD investment, but expect more manual scripting

Implementing Port.io Self-Service ArgoCD Deployments via Orkes and Vault

1 Executive Summary

Goal: Enable developers to **self-service deploy, sync, and rollback ArgoCD applications** securely and efficiently via Port.io.

Proposed Solution:

- **Port.io** triggers workflows.
- **Orkes** handles workflow orchestration (sync, rollback, deploy revisions).
- **Vault** securely manages ArgoCD admin tokens.
- **Kubernetes CSI** mounts secrets into Orkes workers.

Key Benefits:

- Secure secret management.
- Simplified developer experience.
- Reliable and scalable workflows.
- Full observability and auditability.

2 Why Orkes + Vault + Port.io?

Feature	Orkes Approach	Jenkins Alternative
Async workflows	✅ Native support	⚠ Manual scripting
Secret handling	✅ Vault CSI mounted per workflow	⚠ Plugin-based, more manual
Port.io integration	✅ Direct HTTP workflow triggers	⚠ Webhook integration required
Scalability	✅ Parallel tasks, retries, fan-out/fan-in	⚠ Limited by agent scaling
Maintenance	✅ Declarative JSON workflows	⚠ Scripts can drift over time
Security	✅ Least privilege, dynamic tokens	⚠ Needs careful credential management

Conclusion: Orkes + Vault offers **better security, scalability, maintainability, and developer experience** than a Jenkins-centric approach.

3 Requirements

Technical Prerequisites:

- Kubernetes cluster with `secrets-store-csi-driver` installed.
- Vault server deployed and accessible.
- Port.io instance for self-service UI.
- Orkes Conductor server deployed for workflow orchestration.
- ArgoCD server deployed for GitOps application management.

Roles and Access:

- ServiceAccount for Orkes with Vault read-only access.
- Port.io API token for workflow triggers.
- Vault role bound to Orkes ServiceAccount.

4 High-Level Architecture

```
Developer
|
▼
Port.io UI → HTTP Trigger
|
▼
Orkes Worker (workflow: sync/rollback/deploy)
|
├─ Reads ArgoCD token from Vault via CSI
|
▼
ArgoCD API (Sync / Rollback / Deploy Revision)
|
▼
Workflow status returned to Port.io
```

Key Points:

- Secrets never stored in plaintext.
- Developers click a button, workflows handle complex orchestration.
- Workflow outputs (success/failure) shown directly in Port.io.

5 Step-by-Step Implementation Plan

Phase 1: Vault Secret Management

2820. Create Vault policy for ArgoCD tokens.
2821. Enable Kubernetes auth in Vault.
2822. Create Vault role for Orkes ServiceAccount.
2823. Store ArgoCD admin token in Vault.

Phase 2: Kubernetes Setup

3022. Create `orkes` namespace.
3023. Create ServiceAccount `orkes-sa` .
3024. Configure RBAC for reading secrets.
3025. Deploy SecretProviderClass (Vault CSI) for ArgoCD token.

Phase 3: Orkes Worker Deployment

3233. Deploy Orkes worker pod mounting Vault secret via CSI.
3234. Configure environment variables pointing to token location.

Phase 4: Workflow Definitions in Orkes

3404. Define `sync`, `rollback`, and `deploy revision` workflows.
3405. Configure HTTP tasks to call ArgoCD API.
3406. Test workflows independently.

Phase 5: Port.io Integration

3584. Define self-service actions for sync, rollback, deploy revision.
3585. Configure HTTP triggers to Orkes API with Port.io API token.
3586. Test end-to-end from developer click → Orkes → ArgoCD → Port.io feedback.

6 Step-by-Step Developer Experience

3849. Developer clicks **"Sync"** in Port.io.
3850. Port.io sends HTTP request to Orkes workflow.
3851. Orkes workflow fetches ArgoCD token from Vault CSI.
3852. Orkes calls ArgoCD API to sync the application.
3853. Status is returned to Orkes → Port.io updates the UI.
3854. Developer sees **success/failure** instantly.

7 Benefits of This Approach

Category	Benefit
Security	No hard-coded tokens; Vault + CSI for dynamic secrets
Developer Productivity	Click-button self-service → faster deploys
Reliability	Orkes retries, fan-out, error handling
Auditability	Workflow logs and Port.io action tracking
Scalability	Parallel workflows, multiple apps, multi-team
Maintainability	Declarative workflows, minimal scripting

8 Deliverables

- Vault setup scripts and policies.
- Kubernetes manifests (Namespace, SA, RBAC, SecretProviderClass, Orkes worker).
- Orkes workflow JSON for sync, rollback, deploy revision.
- Port.io actions YAML for self-service buttons.
- Documentation: developer guide + admin setup.

9 Future Enhancements

- **Add Git-based approvals**: require PR merge before deploy.
- **Multi-environment support**: staging, QA, prod workflows.
- **Audit dashboards**: central logs from Orkes → Port.io → ArgoCD.
- **Secrets rotation automation** via Vault dynamic tokens.

10 Convincing Reason to Adopt

- Provides **secure, scalable, and maintainable self-service GitOps workflows**.
- Reduces errors by standardizing deployment pipelines.
- Improves developer velocity with **one-click operations**.
- Keeps secrets secure with Vault and Kubernetes CSI.
- Future-proof: Orkes allows extending workflows with additional tasks (notifications, compliance checks, etc.).

✅ **Conclusion:**

By adopting **Port.io + Orkes + Vault + ArgoCD**, the organization enables **secure, reliable, and scalable self-service GitOps**, reducing operational overhead, improving developer experience, and supporting enterprise-grade deployment practices.

Feature	Orkes	Jenkins
Async workflows	✅ Native	⚠ Manual
Secret handling	✅ Vault CSI	⚠ Plugin/manual
Port.io integration	✅ Direct	⚠ Webhook
Scalability	✅ Parallel tasks	⚠ Limited
Maintenance	✅ Declarative	⚠ Scripts drift
Security	✅ Dynamic tokens	⚠ Manual