**Blue-Green Deployment SOP for EKS Kubernetes Version Upgrade (Including Workload Migration and Velero Backup)**

Prerequisites:

AWS CLI installed and configured.

Terraform blueprint

kubectl installed.

Helm (if you use Helm for EKS add-ons).

Velero CLI installed.

Steps:

1. **Install Velero:**

Follow the Velero documentation to install Velero on both the existing (Green) and new (Blue) clusters.

Configure Velero with appropriate storage backend credentials.

2**. Back up Configurations with Velero:**

Use Velero to perform a backup of your existing (Green) cluster's configurations and persistent volumes.

bash

velero backup create green-cluster-backup --include-namespaces=<your-application-namespaces>

3. **Create a Duplicate EKS Cluster (Blue):**

Use the EKS blueprint to create a new EKS cluster with the desired Kubernetes version.

Ensure the new cluster has the same configurations, node groups, and networking as the existing cluster.

1. Run the following AWS CLI command to retrieve the appropriate AMI ID. Replace the AWS Region, Kubernetes version, and platform as appropriate. You must be logged into the AWS CLI using an [IAM principal](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_terms-and-concepts.html) that has the ssm:GetParameter IAM permission to retrieve the Amazon EKS optimized AMI metadata.

Command:

aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.29/amazon-linux-2/recommended/image\_id --region region-code --query "Parameter.Value" --output text

1. Given output value should be utilized in the EKS Blueprint.

4. **Deploy Applications and Add-ons to Blue Cluster:**

Deploy your applications and EKS add-ons to the new cluster.

Ensure that configurations are compatible with the new Kubernetes version.  
 Following process should be ideally guided with a runbook.

5. **Migrate Workloads to Blue Cluster:**

Gradually migrate workloads from the existing (Green) cluster to the new (Blue) cluster using tools like kubectl.

Monitor and validate each workload's functionality on the Blue cluster.

6. **Test on the Blue Cluster:**

Validate that applications and EKS add-ons function correctly on the new cluster.

Perform thorough testing, including functional, performance, and integration testing.

7. **Update DNS or Load Balancer to Route Traffic to Blue:**

If your applications use a DNS or a load balancer, update the configurations to route traffic to the new (Blue) cluster gradually.

Monitor traffic to ensure a smooth transition.

8. **Monitor and Validate:**

Monitor the Blue cluster for any issues during the transition.

Validate that applications are handling traffic correctly on the new cluster.

9. **Rollback Plan:**

In case of issues, have a rollback plan ready to quickly redirect traffic back to the original (Green) cluster.

10. **Update Blue Cluster Node Groups:**

If needed, update the node groups in the new (Blue) cluster for scaling or performance improvements.

11. **Update Green Cluster:**

Once satisfied with the Blue cluster, update the existing (Green) cluster to the new Kubernetes version following the same steps.

12. **Restore from Velero Backup (if needed):**

In the event of any issues during the upgrade, use Velero to restore configurations and persistent volumes to the original (Green) cluster.

bash

velero restore create --from-backup green-cluster-backup

**13**. **Monitor and Validate After Full Transition:**

**1. Control Plane Testing:**

#### **a. Kubernetes Version Compatibility:**

* Test the compatibility of your workloads and applications with the target Kubernetes version.
* Check the EKS release notes for any specific considerations.

#### **b. API Server Functionality:**

* Validate the functionality of the Kubernetes API server.
* Execute basic API calls to ensure responsiveness and correctness.

#### **c. Cluster Management Operations:**

* Test common cluster management operations, such as creating and deleting namespaces, deploying sample applications, and scaling deployments.

#### **d. Authentication and Authorization:**

* Verify that authentication and authorization mechanisms are working as expected.
* Test RBAC policies and ensure access controls are correctly configured.

### **2. Data Plane Testing:**

#### **a. Node Health:**

* Monitor the health of individual nodes in the cluster.
* Check for any unhealthy nodes and investigate potential issues.

#### **b. Pod Scheduling:**

* Validate that pods are scheduled correctly across nodes.
* Check for any unscheduled pods and troubleshoot scheduling issues.

#### **c. Network Connectivity:**

* Test network connectivity between pods and services.
* Verify that services are reachable and communication between pods is functional.

#### **d. Resource Utilization:**

* Monitor resource utilization (CPU, memory, etc.) on nodes.
* Ensure that workloads are appropriately distributed and resource limits are respected.

### **3. Kubernetes Add-ons Testing:**

#### **a. CoreDNS and Kube-Proxy:**

* Validate the functionality of CoreDNS and Kube-Proxy, critical components for DNS resolution and service networking.

#### **b. Dashboard and Metrics Server:**

* Test the Kubernetes dashboard and metrics server for monitoring cluster metrics and health.

#### **c. Ingress Controllers:**

* Validate the functionality of any Ingress controllers used in the cluster.
* Test routing rules and SSL termination.

### **4. Customer Controllers and Custom Resources Testing:**

#### **a. Custom Resource Definitions (CRDs):**

* If you use custom resources, test the creation and deletion of resources.
* Ensure that your custom controllers are reconciling the state correctly.

#### **b. Operator Functionality:**

* If you use operators to manage applications, validate their functionality.
* Test deployment, scaling, and updates of operator-managed resources.

### **5. Kubernetes Manifest Files Testing:**

#### **a. Configuration Validation:**

* Use tools like kubeval to validate Kubernetes manifest files for syntax errors and adherence to the Kubernetes API schema.

#### **b. Resource Deployment:**

* Deploy resources using kubectl apply and verify that the desired state is achieved.

#### **c. Rolling Updates:**

* Test rolling updates for deployments to ensure minimal downtime during updates.

### **6. End-to-End Testing:**

#### **a. Application Workflows:**

* Conduct end-to-end testing of critical application workflows.
* Validate data integrity and correctness of application behavior.

#### **b. Failure Scenarios:**

* Simulate failures, such as node failures, and verify that the system responds gracefully.

#### **c. Scale Testing:**

* Perform scale testing to ensure the cluster can handle increased loads.
* Monitor performance metrics and identify any scaling bottlenecks.

### **7. Monitoring and Observability:**

#### **a. Logging and Monitoring Tools:**

* Verify that logging and monitoring tools (e.g., Prometheus, Grafana, CloudWatch) are collecting and displaying relevant data.

#### **b. Alerts and Notifications:**

* Set up alerts for critical events and test their effectiveness.
* Ensure that relevant team members receive notifications for incidents.

### **8. Security Testing:**

#### **a. Pod Security Policies:**

* If using Pod Security Policies, validate that they are enforced.
* Test the creation of pods to ensure compliance with security policies.

#### **b. Network Policies:**

* If using Network Policies, validate that they restrict traffic according to defined rules.
* Test communication between pods to ensure adherence to network policies.

### **9. Documentation Review:**

#### **a. Operational Documentation:**

* Review and update operational documentation based on testing outcomes.
* Ensure that documentation reflects the current state of the cluster and provides troubleshooting guidance.

14. **Cleanup:**

Once confident in the new setup, decommission the old (Green) cluster.

Clean up any unused resources.