Deploying a Kubernetes cluster with Kubespray involves **hundreds of Ansible tasks**, organized into modular roles and playbooks. It’s not a fixed number, since the exact count depends on your configuration, selected features, and environment (e.g. container runtime, network plugin, OS type, offline mode, etc.).

Here’s a breakdown of the major task categories Kubespray runs during a typical build:

**🧩 Core Task Categories**

* **Inventory setup**: Define hosts, roles (etcd, control plane, nodes), and access parameters.
* **Prechecks**: Validate OS, Python, SSH, swap, SELinux, and firewall settings.
* **Download artifacts**: Pull binaries and container images (or load from local repo in offline mode).
* **Container runtime setup**: Install and configure Docker, containerd, or CRI-O.
* **Etcd deployment**: Configure and initialize the distributed key-value store.
* **Kubernetes control plane**: Deploy kube-apiserver, kube-controller-manager, and kube-scheduler.
* **Kubelet and kube-proxy setup**: Install and configure node agents and networking.
* **Network plugin installation**: Calico, Flannel, Cilium, etc.
* **DNS and ingress setup**: Deploy CoreDNS, NGINX ingress controller, etc.
* **Add-ons**: Optional components like Helm, metrics-server, dashboard, etc.

**🛠️ Task Execution Flow**

Each of these categories contains **dozens of individual tasks**, defined in YAML files under Kubespray’s roles/ directory. For example:

* roles/kubernetes/node/tasks/[main.yml](https://main.yml) includes tasks for kubelet and kube-proxy.
* roles/network\_plugin/calico/tasks/[main.yml](https://main.yml) handles Calico setup.
* roles/download/tasks/download\_file.yml manages binary/image downloads.

**Add-On Integration Workflow**

1. **Enable Add-Ons via Variables**
   * You toggle add-ons in inventory/mycluster/group\_vars/k8s\_cluster/addons.yml.
   * Example:

yaml

dashboard\_enabled: true

helm\_enabled: true

metrics\_server\_enabled: true

ingress\_nginx\_enabled: true

metallb\_enabled: true

1. **Ansible Roles Handle Deployment**
   * Each add-on has a dedicated role under roles/ (e.g. roles/dashboard, roles/helm, etc.).
   * When you run cluster.yml, Kubespray checks which add-ons are enabled and executes the corresponding tasks.
2. **Customization Options**
   * You can configure namespaces, versions, ports, and other settings in the same addons.yml file.
   * For example, MetalLB:

yaml

metallb\_protocol: "layer2"

metallb\_ip\_range:

- "192.168.1.240-192.168.1.250"

1. **Static Manifests or Helm**
   * Most add-ons are deployed using static manifests templated by Kubespray.
   * Some (like Helm itself) can be used post-deployment to install additional components.
2. **Limitations & Considerations**
   * Kubespray uses predefined versions unless overridden.
   * Removing an add-on later requires manual cleanup (e.g. deleting CRDs, RoleBindings, etc.).
   * You can patch or extend Kubespray roles if you need more control over versions or behavior.

**🔍 Example: Enabling NGINX Ingress**

yaml

ingress\_nginx\_enabled: true

ingress\_nginx\_namespace: "ingress-nginx"

ingress\_nginx\_insecure\_port: 80

ingress\_nginx\_secure\_port: 443

This will deploy the ingress controller as a DaemonSet and expose it via a LoadBalancer IP if MetalLB is also enabled.

When it comes to Kubespray, you’ve got a buffet of optional components you can enable during cluster provisioning — perfect for tailoring your setup to your needs, Jason. Here's a curated list of **add-ons and integrations** you can include:

**🚀 Core Add-Ons (Toggle in addons.yml)**

These are officially supported and easily enabled via variables:

| **Add-On** | **Purpose** |
| --- | --- |
| dashboard\_enabled | Kubernetes Dashboard UI |
| helm\_enabled | Helm package manager |
| metrics\_server\_enabled | Resource metrics for HPA and Dashboard |
| ingress\_nginx\_enabled | NGINX Ingress Controller |
| metallb\_enabled | LoadBalancer support for bare metal |
| local\_volume\_provisioner\_enabled | Dynamic local PV provisioning |
| kube\_vip\_enabled | Virtual IP for HA control plane |
| cert\_manager\_enabled | Automated TLS certificate management |
| rook\_enabled | Ceph-based storage provisioning |
| nfs\_provisioner\_enabled | NFS dynamic provisioning |
| kubernetes\_audit\_enabled | Audit logging for API server |

**🧪 Experimental / Advanced Options**

| **Add-On** | **Notes** |
| --- | --- |
| krew\_enabled | Plugin manager for kubectl |
| kube\_ovn\_enabled | Alternative CNI with advanced networking |
| kube\_prometheus\_enabled | Prometheus + Grafana monitoring stack |
| efk\_enabled | Elasticsearch, Fluentd, Kibana logging stack |
| velero\_enabled | Backup and restore for Kubernetes resources |
| istio\_enabled | Service mesh (may require manual tuning) |

**🛠️ How to Enable Them**

In inventory/mycluster/group\_vars/k8s\_cluster/addons.yml, just flip the switch:

yaml

helm\_enabled: true

metrics\_server\_enabled: true

kube\_prometheus\_enabled: true

Some add-ons (like rook or efk) may require additional configuration in their respective group vars files or roles.

**📦 Offline Considerations**

Since you're building offline:

* You'll need to **pre-download container images** for each enabled add-on.
* Some may require **Helm charts or static manifests** to be served locally.
* I can help you script the image pulls and set up a local registry if needed.

To configure **Prometheus and Grafana** in Kubespray, you’ll be enabling the kube-prometheus stack, which includes Prometheus, Grafana, Alertmanager, and node exporters — all wired up for Kubernetes monitoring. Here's how to get it working, especially with your offline setup in mind:

**⚙️ Step 1: Enable the Monitoring Stack**

In your inventory directory, open group\_vars/k8s\_cluster/addons.yml and set:

yaml

kube\_prometheus\_enabled: true

This activates the full monitoring stack during the cluster.yml playbook run.

**🧩 Step 2: Customize Configuration (Optional but Recommended)**

You can fine-tune settings in group\_vars/k8s\_cluster/kube-prometheus.yml. For example:

yaml

kube\_prometheus\_namespace: monitoring

kube\_prometheus\_prometheus\_retention: "15d"

kube\_prometheus\_alertmanager\_enabled: true

kube\_prometheus\_grafana\_admin\_user: admin

kube\_prometheus\_grafana\_admin\_password: supersecure

You can also pre-load custom Grafana dashboards or Prometheus alert rules by placing them in the appropriate roles/kubernetes/kube-prometheus/files/ subdirectories.

**📦 Step 3: Prepare for Offline Deployment**

Since you're building offline, you'll need to:

* **Pre-download container images**:
  + prom/prometheus
  + grafana/grafana
  + quay.io/prometheus/alertmanager
  + quay.io/prometheus/node-exporter
  + quay.io/coreos/kube-state-metrics
  + grafana/grafana-image-renderer (optional)
* **Save and load images**:

bash

docker pull <image>

docker save <image> -o <image>.tar

* **Push to your local registry** or load them directly on each node.

Update your image repo in k8s-cluster.yml:

yaml

kube\_image\_repo: "registry.local:5000"

**🚀 Step 4: Deploy the Cluster**

Run the playbook:

bash

ansible-playbook -i inventory/mycluster/hosts.yaml cluster.yml

Kubespray will deploy Prometheus, Grafana, and supporting components into the monitoring namespace.

**📊 Step 5: Access Grafana**

Once deployed:

* Port-forward Grafana:

bash

kubectl port-forward svc/grafana -n monitoring 3000:3000

* Visit http://localhost:3000
* Login with the credentials you set earlier

Grafana will already have dashboards for Kubernetes nodes, pods, and cluster health.