Comprehensive Guide to Installing Rancher on a Single-Node RKE2 Cluster with Essential Services

1. Introduction

This report provides a comprehensive, step-by-step plan for installing Rancher Management Server on a single-node RKE2 (Rancher Kubernetes Engine 2) cluster. The installation includes several essential components to create a fully functional environment: MetalLB for bare-metal load balancing, Local Path Provisioner for persistent storage, Traefik as the Ingress controller, and Cert-Manager for automated TLS certificate management via Let's Encrypt.
This guide is intended for users with a foundational understanding of Linux server administration and Kubernetes concepts. The target environment is a single server running Ubuntu Server 24.04. Key prerequisites include having DNS configured for the domain rancher.dolboeb.us to point to the public IP address of the router, and port forwarding rules on the router for TCP ports 80 and 443 to the internal IP address of the Ubuntu server where RKE2 will be installed. MetalLB will utilize the IP address range 192.168.1.240-192.168.1.250 for assigning external IPs to services.

2. System and RKE2 Preparation

Before installing RKE2 and Rancher, the underlying Ubuntu system must be properly prepared. This involves meeting hardware requirements, configuring the operating system, and setting up network components.

2.1. Ubuntu System Preparation

2.1.1. Hardware Requirements

Ensure the server meets the minimum hardware requirements for running RKE2 and Rancher. For a small RKE2 deployment, which this single-node setup qualifies as, the recommended minimums are:

- **CPU:** 4 vCPUs
- **RAM:** 16 GB
- **Disk:** SSD is highly recommended for optimal etcd performance, which is critical for RKE2. A minimum of 50 GB of free disk space on the root partition is advisable for the OS, RKE2, container images, and local persistent volumes. RKE2 itself has specific disk requirements, and it's crucial to maintain at least 30% free storage space on the controller node for proper operation.

2.1.2. System Updates and Hostname

First, update the system packages to their latest versions:

```
sudo apt update && sudo apt upgrade -y
sudo reboot # Recommended if kernel or critical packages were updated
```

Verify the system's hostname. While less critical for a single node, RKE2 requires unique hostnames in a multi-node setup unless specific configurations like node-name or with-node-id are used. Check the hostname:

```
hostname
cat /etc/hostname
```

Ensure /etc/hosts correctly maps 127.0.0.1 to localhost and potentially the node's actual IP to its hostname.

2.1.3. Disable Swap

Kubernetes requires swap to be disabled for stable performance and predictable resource management by the kubelet :

```
sudo swapoff -a
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Verify swap is off:

```
sudo swapon --show
free -h
```

The output of sudo swapon --show should be empty, and free -h should show 0B for Swap.

2.1.4. Kernel Modules and System Configuration

Ensure necessary kernel modules for container networking are loaded and persist across reboots:

```
sudo modprobe overlay
sudo modprobe br_netfilter

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter...[source](https://medium.com/@ozcankaraa/kubernetes-dock
er-kurulumlar%C4%B1-2024-k8s-ii-d38c4a79a102)               = 1
EOF

sudo sysctl --system
```

These settings are essential for Kubernetes networking, enabling bridged traffic to be processed by iptables for filtering and port forwarding, and allowing IP forwarding for routing container traffic.

2.1.5. Firewall Configuration

Configure the UFW (Uncomplicated Firewall) to allow necessary traffic. The following table outlines the ports required for this setup:

Port	Protocol	Source	Destination	Purpose	Component	Reference
6443	TCP	Any	Node	Kubernetes API Server	RKE2 Server	
9345	TCP	Any	Node	RKE2 Supervisor API	RKE2 Server	
10250	TCP	Any	Node	Kubelet Metrics	RKE2 Kubelet	
8472	UDP	Other Nodes	Node	CNI Overlay (Flannel VXLAN - RKE2 default CNI is Canal)	RKE2 CNI (Canal)	
7946	TCP/UDP	Other Nodes	Node	MetalLB Memberlist (for L2 mode)	MetalLB	
80	TCP	Router	Node	HTTP Ingress Traffic	Traefik via MetalLB	User Query
443	TCP	Router	Node	HTTPS Ingress Traffic	Traefik via MetalLB	User Query
(Note: etcd ports 2379-2380 are typically bound to localhost on a single-node RKE2 server and may not require explicit firewall opening for external access.)						

Apply these rules using ufw:

```
sudo ufw allow 6443/tcp comment 'RKE2 API Server'
sudo ufw allow 9345/tcp comment 'RKE2 Supervisor API'
sudo ufw allow 10250/tcp comment 'Kubelet Metrics'
sudo ufw allow 8472/udp comment 'RKE2 CNI Flannel VXLAN (Canal)'
sudo ufw allow 7946/tcp comment 'MetalLB Memberlist TCP'
sudo ufw allow 7946/udp comment 'MetalLB Memberlist UDP'
sudo ufw allow 80/tcp comment 'Ingress HTTP (from Router)'
sudo ufw allow 443/tcp comment 'Ingress HTTPS (from Router)'
```

```
# Enable UFW
sudo ufw enable

# Check status
sudo ufw status verbose
```

Ensure your router is configured to forward external ports 80 and 443 to this node's IP address on ports 80 and 443 respectively, as specified in the user query.

2.1.6. NetworkManager Configuration (If Applicable)

If NetworkManager is installed and enabled, it must be configured to ignore CNI-managed interfaces to prevent interference with Kubernetes networking. Create a configuration file:

```
sudo nano /etc/NetworkManager/conf.d/rke2-cni.conf
```

Add the following content:

```
[keyfile]
unmanaged-devices=interface-name:cni0;interface-name:flannel.1;interfa
ce-name:veth*
```

Reload NetworkManager:

```
sudo systemctl reload NetworkManager
```

This step is critical as NetworkManager can otherwise disrupt pod networking by attempting to manage interfaces created by the CNI plugin.

2.2. Install RKE2 Server

RKE2 can be installed using a simple shell script. The INSTALL_RKE2_TYPE="server" variable ensures that the server components are installed.

```
curl -sfL https://get.rke2.io | sudo INSTALL_RKE2_TYPE="server" sh -
```

This command downloads the RKE2 installation script and executes it. The script installs the rke2-server binary and systemd service unit.

2.3. RKE2 Configuration for Single-Node

RKE2 is configured via a YAML file located at /etc/rancher/rke2/config.yaml. This file must be created before starting the rke2-server service for the first time.
Create the configuration directory:

```
sudo mkdir -p /etc/rancher/rke2/
```

Create and edit the RKE2 configuration file:

```
sudo nano /etc/rancher/rke2/config.yaml
```

Add the following content. **Replace <REPLACE_WITH_NODE_IP> with the actual IP address of your Ubuntu server that your router forwards ports 80 and 443 to.**

```
tls-san:
  - "<REPLACE_WITH_NODE_IP>"  # This node's IP address
  - "rancher.dolboeb.us"      # FQDN for Rancher
disable:
  - rke2-ingress-nginx        # Disable default Nginx to use Traefik
write-kubeconfig-mode: "0600" # Secure permissions for kubeconfig
# By default, RKE2 server nodes are schedulable.
# For a single-node cluster, this is desired.
# No specific node-taint needed here to make it schedulable initially.
# We will remove any default control-plane taints later using kubectl.
```

Configuration Details:

Parameter	Value	Description	Reference
tls-san	- "<NODE_IP>"<br>- "rancher.dolboeb.us"	Adds Subject Alternative Names to the Kubernetes API server's TLS certificate. Essential for secure access via IP and FQDN.	
disable	- rke2-ingress-nginx	Disables the RKE2-packaged Nginx ingress controller, preventing port conflicts with the planned Traefik installation.	
write-kubeconfig-mode	"0600"	Sets file permissions for the generated kubeconfig file (/etc/rancher/rke2/rke2.yaml) to be readable/writable only by the owner.	

Disabling rke2-ingress-nginx is crucial because both it and Traefik attempt to use ports 80 and 443 for HTTP/HTTPS traffic. Running both would lead to port binding conflicts. This configuration ensures that only Traefik will manage ingress traffic on these ports.

2.4. Starting and Verifying RKE2

Enable and start the rke2-server service:

```
sudo systemctl enable --now rke2-server.service
```

This command enables the service to start on boot and starts it immediately. The initial startup can take several minutes as RKE2 bootstraps the Kubernetes control plane.
Monitor the startup logs:

```
sudo journalctl -u rke2-server -f
```

Look for messages indicating that etcd, the Kubernetes API server, scheduler, and controller manager have started successfully. You should eventually see messages confirming the node is ready. Press Ctrl+C to exit log monitoring.

2.5. Setting up kubectl

RKE2 installs kubectl (the Kubernetes command-line tool) at /var/lib/rancher/rke2/bin/kubectl. To use it conveniently, add its location to your PATH and set the KUBECONFIG environment variable to point to the RKE2-generated kubeconfig file.
For the current user:

```
echo 'export PATH=$PATH:/var/lib/rancher/rke2/bin' >> ~/.bashrc
echo 'export KUBECONFIG=/etc/rancher/rke2/rke2.yaml' >> ~/.bashrc
source ~/.bashrc
```

This ensures that kubectl commands are directed to your RKE2 cluster. Alternatively, for system-wide access or for the root user, you can create a symbolic link and copy the kubeconfig:

```
# sudo ln -s /var/lib/rancher/rke2/bin/kubectl /usr/local/bin/kubectl
# sudo mkdir -p $HOME/.kube
# sudo cp /etc/rancher/rke2/rke2.yaml $HOME/.kube/config
# sudo chown $(id -u):$(id -g) $HOME/.kube/config
# export KUBECONFIG=$HOME/.kube/config # and add to.bashrc
```

The .bashrc method is generally preferred for user-specific configuration.
After setting up kubectl, verify the cluster and node status:

```
kubectl get nodes
kubectl get pods -A
```

The output of kubectl get nodes should show one node with the STATUS as Ready. The output of kubectl get pods -A should show core Kubernetes components (like coredns, etcd, kube-apiserver) running in the kube-system namespace. The rke2-ingress-nginx pods should *not* be present if it was correctly disabled in the config.yaml.

2.6. Making the Single Node Schedulable

In a single-node Kubernetes cluster, the master node must also function as a worker node, meaning it needs to be schedulable for user workloads, including Rancher itself. RKE2 server nodes are schedulable by default. However, Kubernetes often applies a taint (e.g., node-role.kubernetes.io/control-plane:NoSchedule or node-role.kubernetes.io/master:NoSchedule) to control-plane nodes to prevent arbitrary pods from being scheduled on them. For a single-node setup, this taint must be removed.
Remove the common control-plane taint:

```
kubectl taint nodes --all node-role.kubernetes.io/control-plane-
kubectl taint nodes --all node-role.kubernetes.io/master- # Also
remove this common alternative
```

The hyphen at the end of the taint key (control-plane-) signifies removal of any taint with that key, regardless of its value or effect.
Verify taints on the node:

```
kubectl describe node $(kubectl get nodes -o
jsonpath='{.items.metadata.name}') | grep Taints
```

The output should ideally be Taints: <none> or not show any NoSchedule or NoExecute taints that would prevent workloads.
The user query also included kubectl label node boss node.kubernetes.io/exclude-from-external-load-balancers-. This label is not a standard Kubernetes label nor is it typically used by MetalLB or Traefik for their core functionality. MetalLB uses its own speaker pods to announce service IPs, and RKE2's bundled ServiceLB (Klipper, which is not being used here) has different mechanisms. This labeling step is likely unnecessary for the current setup and can be omitted.

3. Phase 2: Essential Cluster Services

With RKE2 running, the next step is to install essential services for load balancing and persistent storage.

3.1. MetalLB Installation and Configuration

MetalLB provides a network load-balancer implementation for bare-metal Kubernetes clusters, which is necessary for exposing services like Traefik externally. The latest stable version of MetalLB at the time of this writing is v0.14.9.
Install MetalLB using its official manifest:

```
kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.9/config/manif
ests/metallb-native.yaml
```

This command deploys MetalLB components (controller and speaker) into the metallb-system namespace and installs the required CustomResourceDefinitions (CRDs).
Wait for MetalLB pods to be ready:

```
kubectl get pods -n metallb-system -w
```

Ensure the controller and speaker pods are in the Running state. On a single-node cluster, there will be one speaker pod.
Next, configure MetalLB with an IP address pool and an L2 advertisement. Create a file named metallb-config.yaml:

```
sudo nano metallb-config.yaml
```

Add the following content, which defines the IP range 192.168.1.240-192.168.1.250 as requested:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: default-pool
  namespace: metallb-system
spec:
  addresses:
    - 192.168.1.240-192.168.1.250
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: default-l2
  namespace: metallb-system
spec:
  ipAddressPools:
    - default-pool
  # Optional: If MetalLB has trouble determining the correct network
interface for L2 announcements,
  # you can specify it explicitly. Replace 'eth0' with your node's
primary network interface name.
  # interfaces:
  #  - eth0
```

This configuration creates an IPAddressPool named default-pool with the specified IP range and an L2Advertisement named default-l2 that instructs MetalLB to announce these IPs on the local network using Layer 2 protocols (ARP/NDP).
Apply the MetalLB configuration:

```
kubectl apply -f metallb-config.yaml
```

This IP address pool configuration must be applied *after* MetalLB's components are running but *before* services of type LoadBalancer (like Traefik's service) are created. Otherwise, those services would remain in a <pending> state for their external IP.

3.2. Local Path Provisioner Installation

For persistent storage on the single node, Rancher's Local Path Provisioner is a straightforward solution. It dynamically provisions PersistentVolumes using host paths. The latest stable version is v0.0.31.
Install Local Path Provisioner using its manifest:

```
kubectl apply -f
https://raw.githubusercontent.com/rancher/local-path-provisioner/v0.0.
31/deploy/local-path-storage.yaml
```

This deploys the provisioner into the local-path-storage namespace and creates a StorageClass named local-path.
Verify the provisioner pod:

```
kubectl get pods -n local-path-storage -w
```

Wait for the local-path-provisioner pod to be Running.
The local-path-storage.yaml manifest creates the local-path StorageClass but does not mark it as the default. To simplify subsequent installations (like Rancher) that require persistent storage, it's beneficial to set local-path as the default StorageClass. This allows PersistentVolumeClaims to automatically use it without explicitly specifying storageClassName: local-path.
Make local-path the default StorageClass:

```
kubectl patch storageclass local-path -p '{"metadata":
{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}
}'
```

Verify the change:

```
kubectl get sc
```

The local-path StorageClass should now have (default) appended to its name in the output.

4. Phase 3: Ingress and Certificate Management

With networking and storage fundamentals in place, the next phase involves setting up Cert-Manager for TLS certificates and Traefik as the Ingress controller.

4.1. Cert-Manager Installation

Cert-Manager automates the management and issuance of TLS certificates from various sources, including Let's Encrypt. The latest stable version of Cert-Manager is v1.17.2, which is compatible with Kubernetes versions 1.29-1.32.
Add the Jetstack Helm repository, which is the official source for the Cert-Manager chart :

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
```

Install Cert-Manager using Helm. The crds.enabled=true flag ensures that the necessary CustomResourceDefinitions are installed along with Cert-Manager. It's installed into its own cert-manager namespace. The --wait flag ensures the installation completes and components are ready before proceeding.

```
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --version v1.17.2 \
  --set crds.enabled=true \
  --wait
```

Verify Cert-Manager pods are running:

```
kubectl get pods -n cert-manager -w
```

Wait for the cert-manager, cert-manager-cainjector, and cert-manager-webhook pods to be in the Running state. The webhook pod might take a bit longer to become ready as it needs to provision its TLS assets.

4.2. Create Let's Encrypt ClusterIssuer

A ClusterIssuer is a Cert-Manager resource that represents a certificate authority from which to obtain signed certificates. It is cluster-scoped, meaning it can be used to issue certificates in any namespace. This guide uses Let's Encrypt with an HTTP01 challenge.
Create a file named clusterissuer.yaml:

```
sudo nano clusterissuer.yaml
```

Add the following content. **Replace <YOUR_EMAIL_ADDRESS> with your actual email address.** This email is used by Let's Encrypt for registration and important notifications.

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory # Let's
Encrypt Production Server
```

```
    email: "<YOUR_EMAIL_ADDRESS>"                            # Your email
for ACME registration
    privateKeySecretRef:
      name: letsencrypt-prod-account-key                     # Secret to
store ACME account private key
    solvers:
      - http01:
          ingress:
            ingressClassName: traefik                        # Use
Traefik for solving HTTP01 challenges
```

This configuration defines a ClusterIssuer named letsencrypt-prod that uses the Let's Encrypt production ACME server. It specifies an email for account registration and a secret name (letsencrypt-prod-account-key) where Cert-Manager will store the ACME account's private key. The HTTP01 solver is configured to use Ingress resources with the ingressClassName set to traefik. This means Cert-Manager will create temporary Ingress resources during the challenge process, and these Ingresses will be handled by Traefik.
Apply the ClusterIssuer:

```
kubectl apply -f clusterissuer.yaml
```

Verify the ClusterIssuer status:

```
kubectl get clusterissuer letsencrypt-prod -o yaml
```

Check the status field in the output. After a short while, it should indicate Ready: True.

4.3. Traefik Ingress Controller Installation

Traefik will serve as the Ingress controller, routing external HTTP/S traffic to services within the Kubernetes cluster. It will be exposed externally via a LoadBalancer service provided by MetalLB. The current stable Traefik Helm chart version compatible with Traefik Proxy v3 is approximately v35.4.0 (check Artifact Hub for the absolute latest stable version if necessary).
Add the Traefik Helm repository :

```
helm repo add traefik https://traefik.github.io/charts
helm repo update
```

Create a traefik-values.yaml file to customize the Traefik installation:

```
sudo nano traefik-values.yaml
```

Add the following content:

```
# Service configuration: Expose Traefik via LoadBalancer
service:
  enabled: true
  type: LoadBalancer # MetalLB will assign an IP from its pool
  # loadBalancerIP: "" # Optional: Specify a static IP from MetalLB
pool if needed. Otherwise, MetalLB auto-assigns.
  ports:
    web:
      port: 80         # External port for HTTP
      targetPort: web  # Maps to Traefik's 'web' entryPoint (container
port 8000 by default)
    websecure:
      port: 443        # External port for HTTPS
      targetPort: websecure # Maps to Traefik's 'websecure' entryPoint
(container port 8443 by default)

# Entrypoints configuration (Traefik's listening points)
# Default entryPoints web (container port 8000/http) and websecure
(container port 8443/https) are standard.
# The service.ports above map external 80/443 to these internal
entrypoints.
ports:
  web:
    port: 8000        # Traefik container listens on this port for
HTTP
    expose:
      default: true      # Expose this entrypoint through the service
    exposedPort: 80   # Service port (matches service.ports.web.port)
  websecure:
    port: 8443        # Traefik container listens on this port for
HTTPS
    expose:
      default: true      # Expose this entrypoint through the service
    exposedPort: 443  # Service port (matches
service.ports.websecure.port)
    tls:
      enabled: true   # Enable TLS termination on the websecure
entrypoint

# IngressClass configuration: Define 'traefik' as an IngressClass
ingressClass:
  enabled: true
  name: traefik # This name is referenced by cert-manager's
ClusterIssuer and Rancher's Ingress
  isDefaultClass: true # Make this the default IngressClass for
Ingress resources

# Providers: Enable Kubernetes Ingress (for cert-manager challenges)
and CRD (for Traefik IngressRoutes)
providers:
  kubernetesCRD:
    enabled: true # For Traefik's custom resources like IngressRoute,
Middleware
  kubernetesIngress:
    enabled: true # For standard Kubernetes Ingress resources (used by
cert-manager HTTP01)
    # ingressClass: traefik # Optional: if not default, ensure it
```

```
watches for 'traefik' class Ingresses

# Logging configuration
additionalArguments:
  - "--log.level=INFO"    # Set log level (DEBUG, INFO, WARN, ERROR)
  - "--accesslog=false"   # Disable access logs to reduce noise;
enable if needed for debugging
  # To enable access logs:
  # - "--accesslog=true"
  # - "--accesslog.filepath=/var/log/traefik/access.log" # Example
path
  # - "--accesslog.bufferingsize=100"

# RBAC for CRDs (usually enabled by default if CRD provider is
enabled)
rbac:
  enabled: true

# Deployment settings for a single-node cluster
deployment:
  replicas: 1
```

This configuration sets Traefik's service type to LoadBalancer, which MetalLB will manage. It defines the standard HTTP (80) and HTTPS (443) ports. An IngressClass named traefik is created and set as default; this is crucial for Cert-Manager's HTTP01 challenges and for Rancher's Ingress to use Traefik. Both Kubernetes Ingress and CRD providers are enabled. Logging is set to INFO, and access logs are disabled by default but can be enabled. The number of replicas is set to 1, appropriate for a single-node setup.
Install Traefik using Helm (use a recent stable chart version, e.g., v35.4.0):

```
helm install traefik traefik/traefik \
  --namespace traefik \
  --create-namespace \
  --values traefik-values.yaml \
  --version v35.4.0 \
  --wait
```

Verify Traefik pods and service:

```
kubectl get pods -n traefik -w
kubectl get svc -n traefik traefik
```

The traefik service in the traefik namespace should show an EXTERNAL-IP assigned by MetalLB from the 192.168.1.240-192.168.1.250 range. This external IP is where traffic forwarded by your router to the node's IP on ports 80/443 will ultimately be directed by MetalLB.

5. Phase 4: Rancher Server Installation

The final phase is installing the Rancher Management Server.

5.1. Add Rancher Helm Repository

Add the official stable Helm repository for Rancher :

```
helm repo add rancher-stable
https://releases.rancher.com/server-charts/stable
helm repo update
```

5.2. Create cattle-system Namespace

Rancher must be installed in the cattle-system namespace :

```
kubectl create namespace cattle-system
```

5.3. Install Rancher Server via Helm

Create a rancher-values.yaml file for Rancher's Helm chart configuration:

```
sudo nano rancher-values.yaml
```

Add the following content. **Replace <YOUR_EMAIL_ADDRESS> with your actual email and <YOUR_RANCHER_ADMIN_PASSWORD> with a strong, unique password.**

```
hostname: rancher.dolboeb.us

ingress:
  tls:
    source: letsEncrypt # Use Let's Encrypt via cert-manager
  ingressClassName: traefik # Ensure Rancher's Ingress uses the
'traefik' IngressClass

letsEncrypt:
  email: "<YOUR_EMAIL_ADDRESS>" # Required for Let's Encrypt
  # For Let's Encrypt with an external cert-manager and a
pre-configured ClusterIssuer,
  # the 'ingress.class' here for ACME challenges might be handled by
the ClusterIssuer's config.
  # If Rancher were to create its own Certificate resource without
specifying an issuer,
  # it might need this. However, since we have a ClusterIssuer,
Rancher's Certificate
  # object (if it creates one, or if one is created for its Ingress)
should reference it.
  # The 'ingress.ingressClassName: traefik' on the main Ingress is
key.

# privateCA: true
# Rancher docs suggest privateCA=true for Let's Encrypt if
```

```
agentTlsMode=strict.
# This implies agents need to trust the LE CA chain, which is usually
in system trust stores.
# For simplicity, this is commented out. If agent connection issues
occur,
# investigate agentTlsMode and potentially provide LE's CA via
additionalTrustedCAs.
#
See.[span_76](start_span)[span_76](end_span)[span_77](start_span)[span
_77](end_span)

persistence:
  enabled: true
  storageClass: "local-path" # Use the local-path StorageClass
(default or explicit)
  size: 20Gi                 # Adjust size as needed for Rancher data

replicas: 1 # Suitable for a single-node RKE2 cluster

# 'addLocal' is deprecated. Rancher imports the local RKE2 cluster by
default.
# 'restrictedAdmin: false' (default) gives the admin full rights to
the local cluster.
# Set to true to restrict admin's ability to manage the local cluster
directly.
restrictedAdmin: false

bootstrapPassword: "<YOUR_RANCHER_ADMIN_PASSWORD>" # Set initial admin
password
```

Rancher Helm Values Explanation:

Helm Key	Value	Description	Reference
hostname	rancher.dolboeb.us	FQDN for accessing Rancher UI and API.	
ingress.tls.source	letsEncrypt	Instructs Rancher to obtain TLS certificates from Let's Encrypt using the installed Cert-Manager.	
ingress.ingressClassName	traefik	Specifies that Rancher's main Ingress resource should be managed by the Traefik Ingress controller.	Implied by best practice; ensures correct Ingress handling.
letsEncrypt.email	<YOUR_EMAIL_ADDRESS>	Your email address, required for Let's Encrypt registration and renewal notifications.	
persistence.enabled	true	Enables persistent storage for Rancher data.	(principles from rancher-backup chart)
persistence.storageClass	local-path	Specifies the StorageClass to use. local-path was set as default, but being explicit is good.	(principles from rancher-backup chart)
persistence.size	20Gi	The size of the persistent volume for Rancher.	
replicas	1	Number of Rancher server pods. For a single-node cluster, this must be 1.	
restrictedAdmin	false	Default is false. When false, the Rancher admin has full control over the local Kubernetes cluster where Rancher is installed. addLocal is deprecated.	
bootstrapPassword	<YOUR_SECURE_PASSWORD>	Sets the initial password for the admin user. If not set,	

Rancher generates one. | |
The privateCA: true setting is often mentioned with Let's Encrypt when agentTlsMode is strict (default in newer Rancher versions). This ensures that Rancher agents can validate the Rancher server's certificate. For Let's Encrypt, this usually means the node's system trust store should include Let's Encrypt's root CAs. For this lab setup, it's initially omitted for simplicity, assuming the Ubuntu server trusts common public CAs. If agent connectivity issues arise, this setting, along with additionalTrustedCAs, would be the primary area to investigate.
Install Rancher using Helm. Use the latest stable Rancher version (e.g., v2.11.2, check Artifact Hub for the chart version corresponding to this app version).

```
helm install rancher rancher-stable/rancher \
  --namespace cattle-system \
  --values rancher-values.yaml \
  --version 2.11.2 \
  --wait
```

The --wait flag ensures Helm waits for the Rancher deployment to complete.
Verify Rancher pods are running:

```
kubectl get pods -n cattle-system -w
```

Wait for the rancher pod(s) to be in the Running state.
Verify Rancher Ingress:

```
kubectl get ingress -n cattle-system
```

Inspect the Ingress resource. It should be using the traefik IngressClass, have rules for rancher.dolboeb.us, and TLS configured with a secret that Cert-Manager will populate.

5.4. Retrieving Bootstrap Password and Accessing Rancher UI

If you set a bootstrapPassword in rancher-values.yaml, use that password. If not, or if you need to retrieve the auto-generated one, use the following command :

```
kubectl get secret --namespace cattle-system bootstrap-secret -o
go-template='{{.data.bootstrapPassword|base64decode}}{{"\n"}}'
```

Alternatively, the password might be found in the Rancher pod logs :

```
kubectl logs -n cattle-system $(kubectl get pods -n cattle-system -l
app=rancher -o jsonpath='{.items.metadata.name}') | grep "Bootstrap
Password:"
```

Access the Rancher UI by navigating to https://rancher.dolboeb.us in your web browser. You should be greeted by the Rancher login page. Log in with the username admin and the bootstrap password you set or retrieved. You will be prompted to set a new password upon first login.

6. Conclusion

This report has detailed the steps to deploy Rancher on a single-node RKE2 Kubernetes cluster running on Ubuntu 24.04. The setup includes essential components: RKE2 as the Kubernetes

distribution, MetalLB for LoadBalancer services, Local Path Provisioner for persistent storage, Traefik as the Ingress controller, and Cert-Manager for automatic TLS certificate management with Let's Encrypt.
The resulting environment provides a fully functional Rancher installation accessible via https://rancher.dolboeb.us, secured with a Let's Encrypt certificate. While this single-node configuration is suitable for development, testing, or lab environments, it does not offer high availability and is not recommended for production workloads that require resilience against node failure.
Further steps would typically involve exploring the Rancher UI, importing or creating additional Kubernetes clusters (if applicable), and deploying applications. This robust single-node setup serves as an excellent platform for learning and experimenting with Rancher and Kubernetes.

Источники

1. Installation Requirements | Rancher, https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/installation-requirements 2. Hardware requirements for RKE2 cluster deployment | Dell Telecom Infrastructure Automation Suite 2.1 Installation and Administration Guide, https://infohub.delltechnologies.com/l/dell-telecom-infrastructure-automation-suite-2-1-installation-and-administration-guide/hardware-requirements-for-rke2-cluster-deployment/ 3. Requirements | RKE2, https://docs.rke2.io/install/requirements 4. Install Rancher RKE2 Cluster three nodes manually - GitHub Gist, https://gist.github.com/devops-school/d9dfa33093499cbde0be46418ee646c4 5. Quick Start | RKE2, https://docs.rke2.io/install/quickstart 6. RKE2 Control-plane Deployment - Expertflow CX, https://docs.expertflow.com/cx/4.4/rke2-single-node-installation-without-ha 7. Setting up a High-availability SUSE® Rancher Prime: RKE2 Kubernetes Cluster for SUSE® Rancher Prime - SUSE Documentation, https://documentation.suse.com/cloudnative/rancher-manager/latest/en/installation-and-upgrade/install-kubernetes/rke2-for-rancher.html 8. High Availability :: Rancher product documentation, https://documentation.suse.com/cloudnative/rke2/latest/en/install/ha.html 9. Advanced Options and Configuration - RKE2, https://docs.rke2.io/advanced 10. Networking Services - RKE2, https://docs.rke2.io/networking/networking_services 11. External datastore - RKE2, https://docs.rke2.io/datastore/external 12. The system-upgrade-controller is not updating tainted nodes | Support - SUSE, https://www.suse.com/support/kb/doc?id=000021800 13. metallb/metallb: A network load-balancer implementation for Kubernetes using standard routing protocols - GitHub, https://github.com/metallb/metallb 14. Releases · metallb/metallb - GitHub, https://github.com/metallb/metallb/releases 15. Installation :: MetalLB, bare metal load-balancer for Kubernetes, https://metallb.universe.tf/installation/ 16. MetalLB Load Balancer - Documentation - k0s docs, https://docs.k0sproject.io/stable/examples/metallb-loadbalancer/ 17. Configuration :: MetalLB, bare metal load-balancer for Kubernetes, https://metallb.universe.tf/configuration/ 18. rancher/local-path-provisioner Tags - Docker Hub, https://hub.docker.com/r/rancher/local-path-provisioner/tags 19. Releases · rancher/local-path-provisioner - GitHub, https://github.com/rancher/local-path-provisioner/releases 20. rancher/local-path-provisioner: Dynamically provisioning ... - GitHub, https://github.com/rancher/local-path-provisioner 21. local-path-provisioner 0.0.32 - Artifact Hub, https://artifacthub.io/packages/helm/containeroo/local-path-provisioner 22. raw.githubusercontent.com,

https://raw.githubusercontent.com/rancher/local-path-provisioner/v0.0.31/deploy/local-path-storage.yaml 23. Helm - cert-manager Documentation, https://cert-manager.io/docs/installation/helm/ 24. Supported Releases - cert-manager Documentation, https://cert-manager.io/docs/releases/ 25. cert-manager 1.17.2 - Artifact Hub, https://artifacthub.io/packages/helm/cert-manager/cert-manager 26. Kubernetes - cert-manager Documentation, https://cert-manager.io/v1.1-docs/installation/kubernetes/ 27. kubectl apply - cert-manager Documentation, https://cert-manager.io/docs/installation/kubectl/ 28. How to Install Traefik Ingress controller with Cert-Manager on Kubernetes | Vultr Docs, https://docs.vultr.com/how-to-install-traefik-ingress-controller-with-cert-manager-on-kubernetes 29. HTTP01 - cert-manager Documentation, https://cert-manager.io/docs/configuration/acme/http01/#configuring-the-http01-ingress-solver 30. traefik 35.4.0 · traefik/traefik - Artifact Hub, https://artifacthub.io/packages/helm/traefik/traefik 31. Traefik Installation Documentation - Traefik, https://doc.traefik.io/traefik/getting-started/install-traefik/ 32. How to deploy Traefik Ingress Controller on Kubernetes using Helm - Platform9 Learning, https://platform9.com/learn/v1.0/tutorials/traefik-ingress 33. Install Traefik Hub Gateway on Kubernetes, https://doc.traefik.io/traefik-hub/api-gateway/setup/kubernetes/installation 34. Traefik Charts | charts, https://helm.traefik.io/traefik 35. traefik-helm-chart/traefik/templates/ingressclass.yaml at master - GitHub, https://github.com/traefik/traefik-helm-chart/blob/master/traefik/templates/ingressclass.yaml 36. Kubernetes IngressRoute & Traefik CRD, https://doc.traefik.io/traefik/providers/kubernetes-crd/ 37. Install/Upgrade Rancher on a Kubernetes Cluster | Rancher, https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/install-upgrade-on-a-kubernetes-cluster 38. Choosing a Rancher Version, https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/resources/choose-a-rancher-version 39. rancher-backup 106.0.2+up7.0.1 · webofmars/rancher-charts - Artifact Hub, https://artifacthub.io/packages/helm/rancher-charts/rancher-backup 40. Rancher Helm Chart Options | Rancher, https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/installation-references/helm-chart-options 41. Rancher on Rancher Desktop, https://docs.rancherdesktop.io/how-to-guides/rancher-on-rancher-desktop/ 42. Setting up the Bootstrap Password - Rancher, https://ranchermanager.docs.rancher.com/getting-started/installation-and-upgrade/resources/bootstrap-password 43. rancher 2.11.2 · helm/rancher-stable - Artifact Hub, https://artifacthub.io/packages/helm/rancher-stable/rancher