Site Reliability Engineer

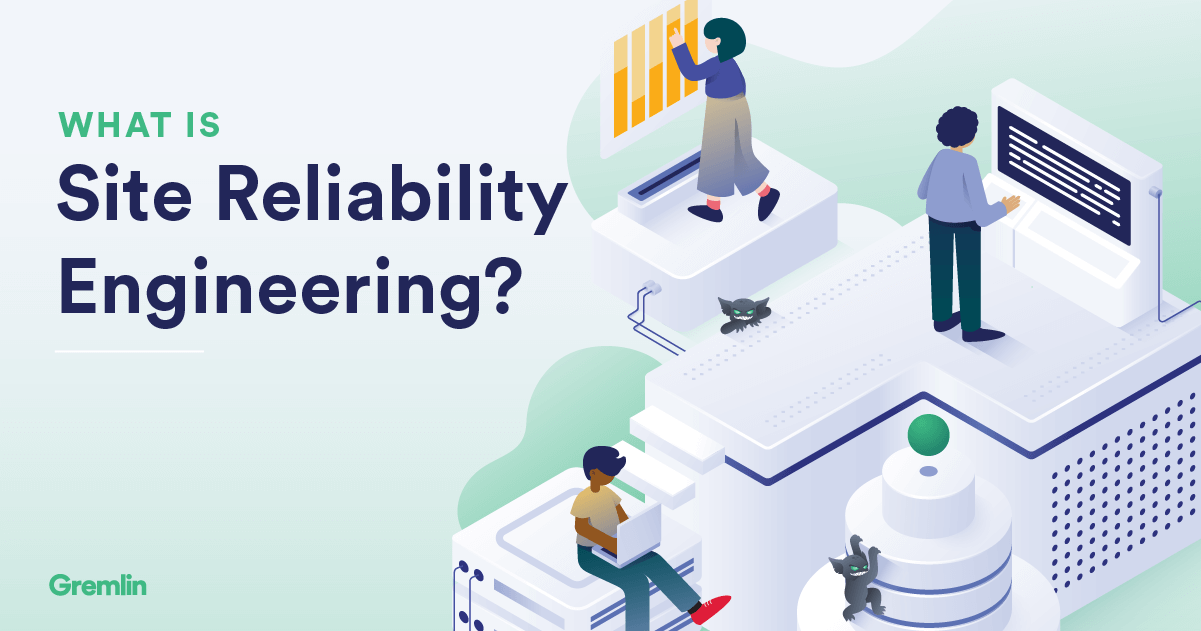


Table of Content

* Introduction
* Operating System – Linux / Ubuntu / CentOS
* Docker and Kubernetes

**Introduction**

*‘Hope is not a strategy.’*

Traditional SRE saying

It is a truth universally acknowledged that systems do not run themselves. How, then, **should** a system—particularly a complex computing system that operates at a large scale—be run?

The Sysadmin Approach to Service Management

Historically, companies have employed systems administrators to run complex computing systems.

This systems administrator, or sysadmin, approach involves assembling existing software components and deploying them to work together to produce a service. Sysadmins are then tasked with running the service and responding to events and updates as they occur.

Google’s Approach to Service Management: Site Reliability Engineering

Conflict isn’t an inevitable part of offering a software service. Google has chosen to run our systems with a different approach: our Site Reliability Engineering teams focus on hiring software engineers to run our products and to create systems to accomplish the work that would otherwise be performed, often manually, by sysadmins.

**DevOps or SRE?**

The term “DevOps” emerged in industry in late 2008 and as of this writing (early 2016) is still in a state of flux. Its core principles—involvement of the IT function in each phase of a system’s design and development, heavy reliance on automation versus human effort, the application of engineering practices and tools to operations tasks—are consistent with many of SRE’s principles and practices. One could view DevOps as a generalization of several core SRE principles to a wider range of organizations, management structures, and personnel. One could equivalently view SRE as a specific implementation of DevOps with some idiosyncratic extensions.

Monitoring

Monitoring is one of the primary means by which service owners keep track of a system’s health and availability. As such, monitoring strategy should be constructed thoughtfully. A classic and common approach to monitoring is to watch for a specific value or condition, and then to trigger an email alert when that value is exceeded or that condition occurs. However, this type of email alerting is not an effective solution: a system that requires a human to read an email and decide whether or not some type of action needs to be taken in response is fundamentally flawed. Monitoring should never require a human to interpret any part of the alerting domain. Instead, software should do the interpreting, and humans should be notified only when they need to take action.

There are three kinds of valid monitoring output:

Alerts

Signify that a human needs to take action immediately in response to something that is either happening or about to happen, in order to improve the situation.

Tickets

Signify that a human needs to take action, but not immediately. The system cannot automatically handle the situation, but if a human takes action in a few days, no damage will result.

Logging

No one needs to look at this information, but it is recorded for diagnostic or forensic purposes. The expectation is that no one reads logs unless something else prompts them to do so.

Emergency Response

Reliability is a function of mean time to failure (MTTF) and mean time to repair (MTTR) [[Sch15]](https://sre.google/sre-book/bibliography#Sch15). The most relevant metric in evaluating the effectiveness of emergency response is how quickly the response team can bring the system back to health—that is, the MTTR.

Change Management

SRE has found that roughly 70% of outages are due to changes in a live system. Best practices in this domain use automation to accomplish the following:

* Implementing progressive rollouts
* Quickly and accurately detecting problems
* Rolling back changes safely when problems arise

Demand Forecasting and Capacity Planning

Demand forecasting and capacity planning can be viewed as ensuring that there is sufficient capacity and redundancy to serve projected future demand with the required availability.

Several steps are mandatory in capacity planning:

* An accurate organic demand forecast, which extends beyond the lead time required for acquiring capacity
* An accurate incorporation of inorganic demand sources into the demand forecast
* Regular load testing of the system to correlate raw capacity(servers, disks, and so on) to service capacity

Because capacity is critical to availability, it naturally follows that the SRE team must be in charge of capacity planning, which means they also must be in charge of provisioning.

Provisioning

Provisioning combines both change management and capacity planning. In our experience, provisioning must be conducted quickly and only when necessary, as capacity is expensive. This exercise must also be done correctly or capacity doesn’t work when needed. Adding new capacity often involves spinning up a new instance or location, making significant modification to existing systems (configuration files, load balancers, networking), and validating that the new capacity performs and delivers correct results. Thus, it is a riskier operation than load shifting, which is often done multiple times per hour, and must be treated with a corresponding degree of extra caution.

Efficiency and Performance

Efficient use of resources is important any time a service cares about money. Because SRE ultimately controls provisioning, it must also be involved in any work on utilization, as utilization is a function of how a given service works and how it is provisioned. It follows that paying close attention to the provisioning strategy for a service, and therefore its utilization, provides a very, very big lever on the service’s total costs.

To become an SRE engineer – you need to go through below mentioned path:

* Learn how to Code
* Acquire in-depth knowledge of version control
* Get knowledge of Operating Systems
* Get familiar with cloud-native applications
* Build understanding of Distributed computing
* Become an expert on CI/CD process
* Acquire in-depth understanding of monitoring tools
* Acquire troubleshooting knowledge
* Be good in communication and buil collaboration skills

**Based on the knowledge requirement, here is the list of required skills and toolsets for SRE engineers**

1. **Operating Systems – Centos/Ubuntu & VirtualBox & Vagrant**
2. **Cloud – AWS**
3. **Containers – Docker & Kubernetes – Helm**

**{We will focus on this above toolset maximum.}**

1. Planning and Designing – Jira & Confluence
2. Source Code Versioning – Git using Github
3. Webserver – Apache HTTP & Nginx
4. Configuration & Deployment Management – Ansible
5. Infrastructure Coding – Terraform
6. Services mesh Data planes & Control Planes – Envoy & Istio
7. Network configurations and Service Discovery – Consul
8. Continuous Integration – Jenkins
9. Securing credentials – HashiCorp Vault & SSL & Certificates
10. Infrastructure Monitoring – Datadog, Prometheus with Grafana
11. Log Monitoring – Splunk & ELK stake
12. Performance & RUM Monitoring – NewRelic
13. Emergency Response & Alerting & Chat & Notification SMTP, SES, SNS,Pagerduty & Slack – Pagerduty & Slack
    1. **Operating System – Linux / Ubuntu / CentOS**

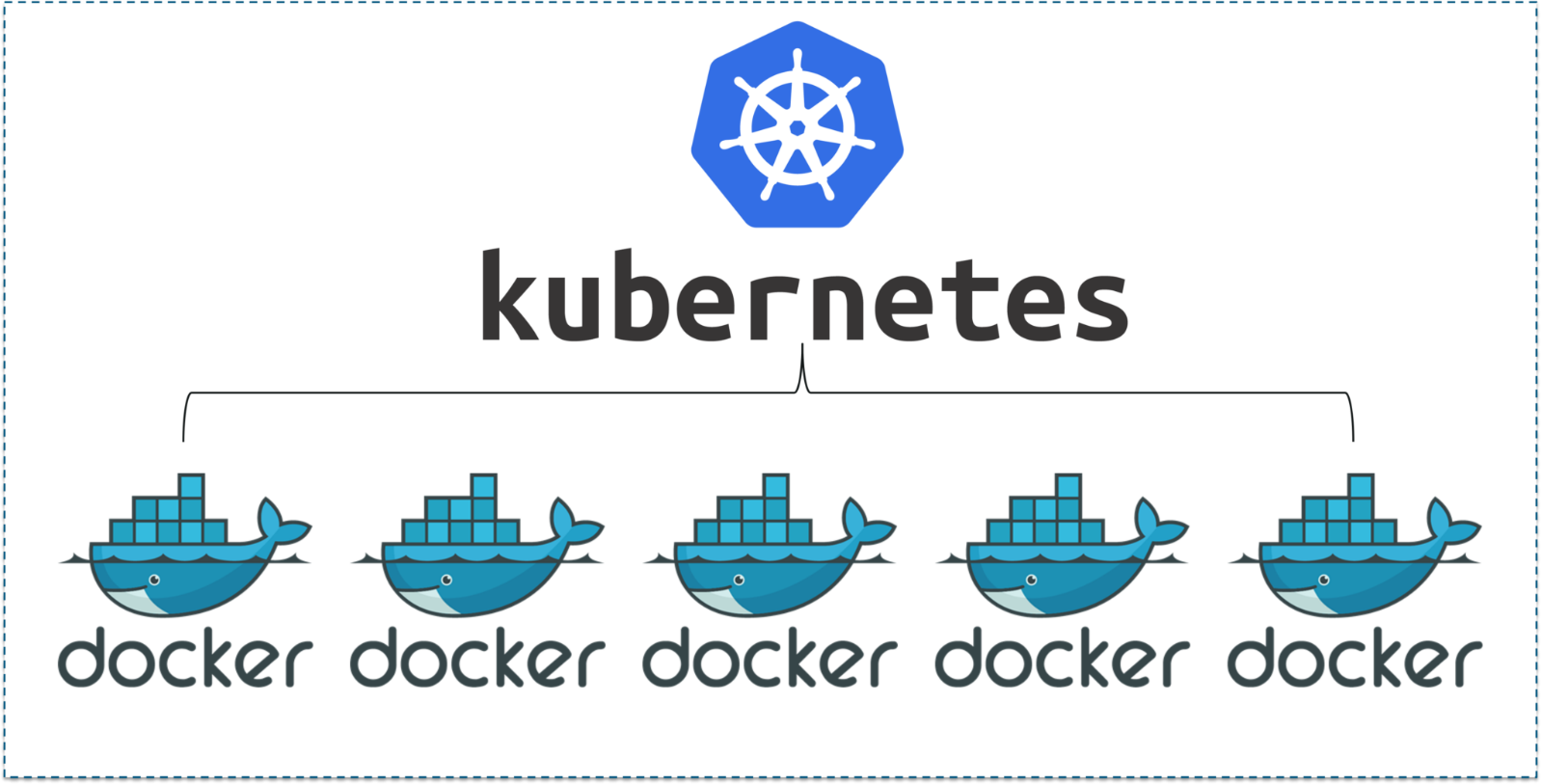
Follow the following command to learn basic commands of linux operating systems.

[Linux/Unix Tutorial - javatpoint](https://www.javatpoint.com/linux-tutorial)

Linux Networking Commands

|  |  |
| --- | --- |
| [ifconfig](https://www.javatpoint.com/linux-ifconfig) | Display and manipulate route and network interfaces. |
| [ip](https://www.javatpoint.com/linux-ip) | It is a replacement of ifconfig command. |
| [traceroute](https://www.javatpoint.com/linux-traceroute) | Network troubleshooting utility. |
| [tracepath](https://www.javatpoint.com/linux-tracepath) | Similar to traceroute but doesn't require root privileges. |
| [ping](https://www.javatpoint.com/linux-ping) | To check connectivity between two nodes. |
| [netstat](https://www.javatpoint.com/linux-netstat) | Display connection information. |
| [ss](https://www.javatpoint.com/linux-ss) | It is a replacement of netstat. |
| [dig](https://www.javatpoint.com/linux-dig) | Query DNS related information. |
| [nslookup](https://www.javatpoint.com/linux-nslookup) | Find DNS related query. |
| [route](https://www.javatpoint.com/linux-route) | Shows and manipulate IP routing table. |
| [host](https://www.javatpoint.com/linux-host) | Performs DNS lookups. |
| [arp](https://www.javatpoint.com/linux-arp) | View or add contents of the kernel's ARP table. |
| [iwconfig](https://www.javatpoint.com/linux-iwconfig) | Used to configure wireless network interface. |
| [hostname](https://www.javatpoint.com/linux-hostname) | To identify a network name. |
| [curl or wget](https://www.javatpoint.com/linux-curl-and-wget) | To download a file from internet. |
| [mtr](https://www.javatpoint.com/linux-mtr) | Combines ping and tracepath into a single command. |
| [whois](https://www.javatpoint.com/linux-whois) | Will tell you about the website's whois. |
| [ifplugstatus](https://www.javatpoint.com/linux-ifplugstatus) | Tells whether a cable is plugged in or not. |

**Docker and Kubernetes**



Kubernetes is the reigning market leader when it comes to container orchestration! Any organization working with the container ecosystem is either already using Kubernetes or considering it. However, despite the undoubted ease and speed Kubernetes bring to the container ecosystem, they also need specialized expertise to deploy and manage.

then managed Kubernetes is the smartest and most cost-effective way ahead. With professionals in the picture, you can be assured of getting long term strategy, seamless implementation, and dedicated on-going service, which will

* reduce deployment time
* provide 24×7 support
* handle all upgrades and fixes
* troubleshoot as and when needed

Kubernetes solution providers offer a wide range of services – from fully managed to bare bone implementation to preconfigured Kubernetes environments on SaaS models to training for your in-house staff.

The first step is to list down all pods after installing your application. The following command lists down all pods in all namespaces.

kubectl get pods -A

##### **Debugging Pods**

###### **Pod Status Shows ImagePullBackOff or ErrImagePull**

This status indicates that your pod could not run because the pod could not pull the image from the container registry. To confirm this, run the kubectl describe command along with the pod identifier to display the details.

kubectl describe pod <pod-identifier>

This command will provide more information about the issue.

* Image name or tag incorrect.
  + Check the image name and tag and try to pull the image manually on the host using docker pull to verify.
* Authentication failure related to Container registry.
  + Check the secrets, roles, service principal related to your container registry and try to pull the image manually on the host using docker pull to verify.

docker pull <image-name:tag>

###### **Pod Status Shows Waiting**

This status indicates your pod has been scheduled to a worker node, but it can’t run on that machine. To confirm this, run the kubectl describe command along with the pod identifier to display the details.

kubectl describe pod <pod-identifier> -n <namespace>

###### **Pod Status Shows Pending or CrashLoopBackOff**

This status indicates your pod could not be scheduled on a node for various reasons like resource constraints (insufficient CPU or memory resources), volume mounting issues.  To confirm this, run the kubectl describe command along with the pod identifier to display the details.

kubectl describe pod <pod-identifier> -n <namespace>

This command will provide more information about the issue. Most common issues are

* Insufficient resources
  + If resources are insufficient, clean up your existing resources or scaling your nodes (vertically or horizontally) to increase the resources.
* Volume mounting
  + Check your volume’s mounting definition and storage classes.
* Using hostPort
  + When you bind a Pod to a hostPort, there are a limited number of places that a pod can be scheduled. In most cases, hostPort is unnecessary, try using a Service object to expose your pod. If you do require hostPort, then you can only schedule as many Pods as there are nodes in your Kubernetes cluster

###### **Pod is crashing or unhealthy**

Sometimes the scheduled pods are crashing or unhealthy.  Run kubectl logs to find the root cause.

kubectl logs <pod\_identifier> -n <namespace>

If you have multiple containers, run the following command to find the root cause.

kubectl logs <pod\_identifier> -c <container\_name> -n <namespace>

If your container has previously crashed, you can access the previous container’s crash log with:

kubectl logs –previous <pod\_identifier> -c <container\_name> -n <namespace>

If your pod is running but with 0/1 ready state or 0/2 ready state (in case if you have multiple containers in your pod), then you need to verify the readiness. Check the health check (readiness probe) in this case.

Most common issues are

* Application issues
  + Run the below command to check the logs.

kubectl logs <pod\_identifier> -c <container\_name> -n <namespace>

* Run the below command to verify the events.

kubectl describe <pod\_identifier> -n <namespace>

* Readiness probe health check failed
  + Check the health check (readiness probe) in this case. Also, check the READY column of the kubectl get pods output to find out if the readiness probe is executing positively.
  + Run the below command to check the logs.

kubectl logs <pod\_identifier> -c <container\_name> -n <namespace>

* Run the below command to verify the events.

kubectl describe <pod\_identifier> -n <namespace>

* Liveness probe health check failed
  + Check the health check (liveness probe) in this case. Also, check the RESTARTS column of the kubectl get pods output. To find out if the liveness probe is executing positively.
  + Run the below command to check the logs.

kubectl logs <pod\_identifier> -c <container\_name> -n <namespace>

* Run the below command to verify the events.

kubectl describe <pod\_identifier> -n <namespace>

###### **Pod is running but has application issues**

In some cases, the pods are running, but the output of the application is incorrect. In this case, you should run the following to find the root cause.

* Run the below command and identify the issue.

kubectl logs <pod\_identifier> -c <container\_name> -n <namespace>

* If you are interested in the last n lines of logs run

kubectl logs <pod\_identifier> -c <container\_name> --tail <n-lines> -n <namespace>

* Run the commands inside the container using

kubectl exec <pod\_identifier> -c <container\_name> /bin/bash -n <namespace>

Run the commands like ‘curl’ or ‘ps’ ‘ls’ to troubleshoot the issue after you get into the container.

###### **Pod is running and working but cannot access through services**

In some cases, the pods are working as expected but cannot access through the services. Most common causes of this issue are

* Service not registered properly
  + Check that the service exists and describe the service and validate the pod selectors to run the following commands.

kubectl get svc

kubectl describe svc <svc-name>

kubectl get endpoints

* Run the following commands to verify pod selector

kubectl get pods --selector=name={name},{label-name}={label-value}

* The service may be deployed in a different namespace.
  + Verify that the pod’s containerPort matches up with the Service’s containerPort
* Service is registered properly but has a DNS issue
  + Get into the container using exec command and run nslookup using the following command

kubectl get endpoints

kubectl exec <pod\_identifier> -c <container\_name> /bin/bash

nslookup <service-name>

* If you have any issues to run the command for curl or nslookup. Deploy debugging pod using image yauritux/busybox-curl in the same namespace to verify. Please run the following commands to verify

kubectl run --generator=run-pod/v1 -it --rm <name> --image=yauritux/busybox-curl -n <namespace>

* Run the following to verify within the container
* telnet <service-ip> <service-port>
* nslookup <servicename>