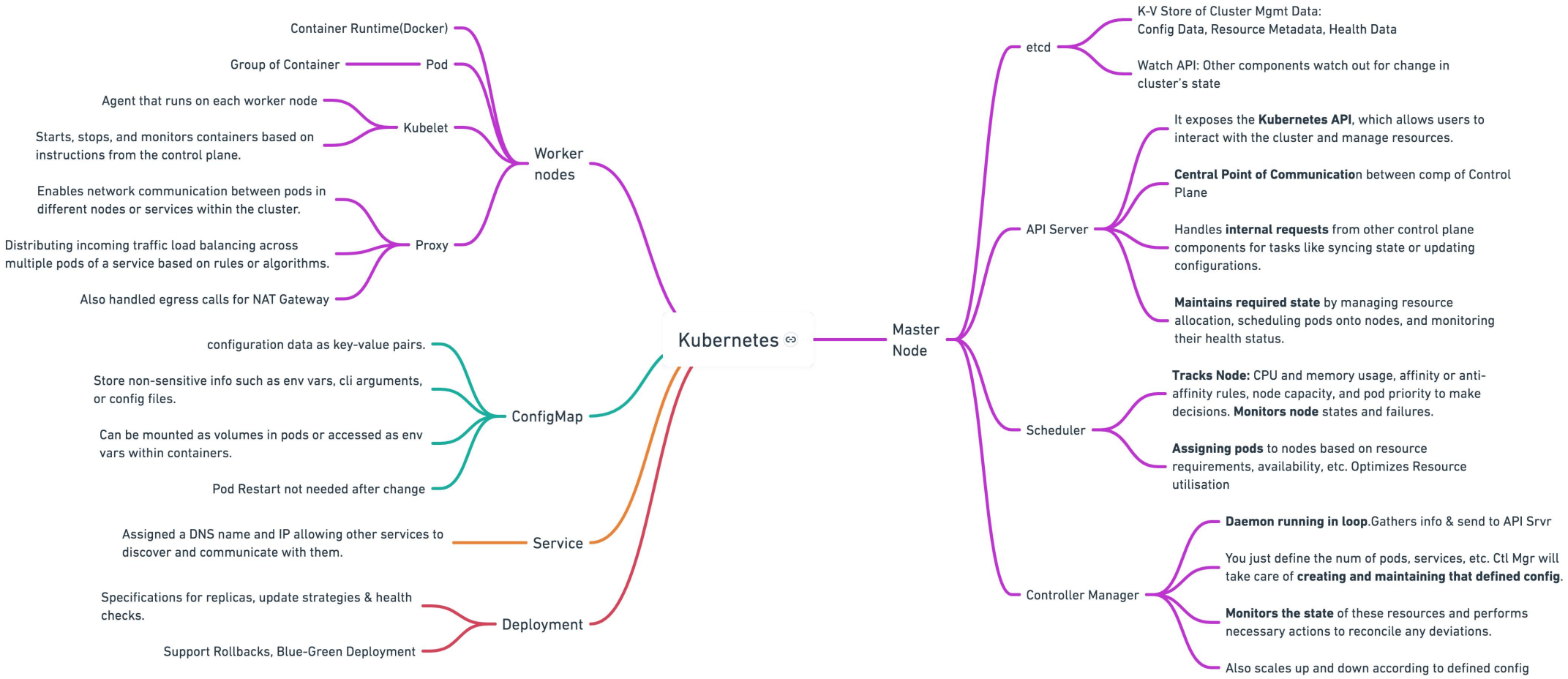
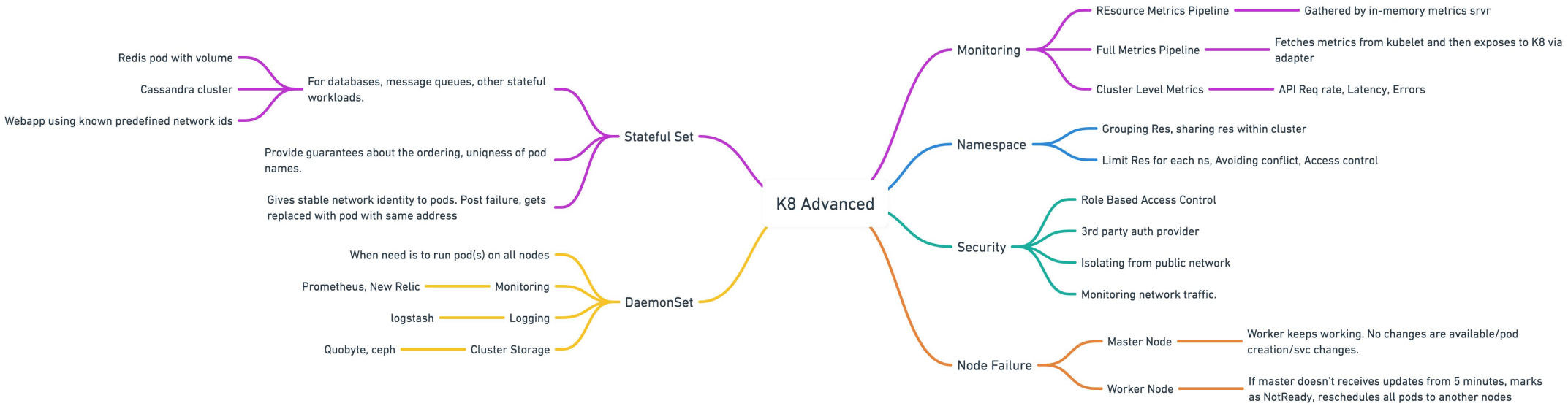
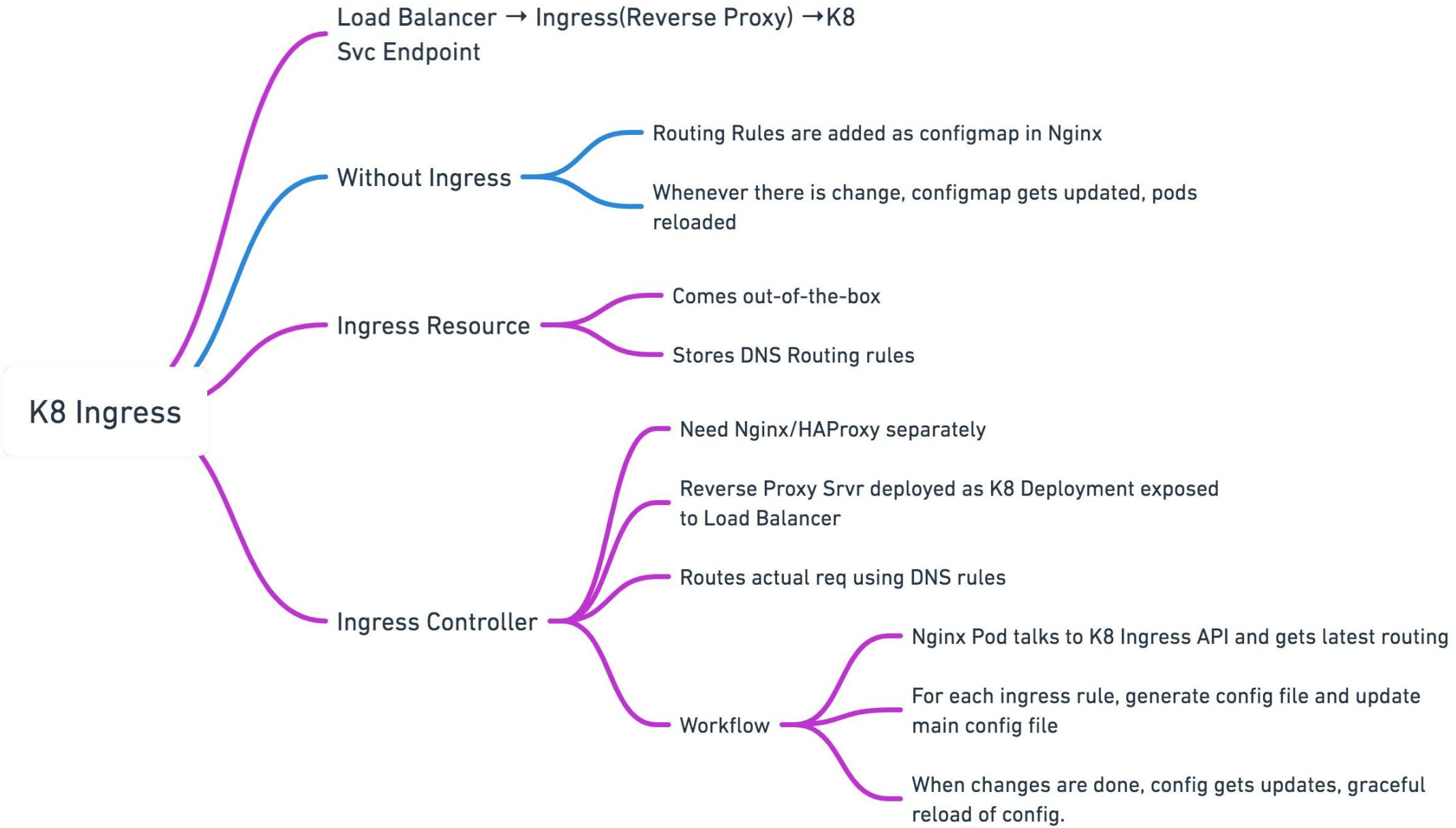

Kubernetes
Worker nodes
Container Runtime(Docker)
Pod
Group of Container
Kubelet
Agent that runs on each worker node
Starts, stops, and monitors containers based on instructions from the control plane.
Proxy
Enables network communication between pods in different nodes or services within the cluster.
Distributing incoming traffic load balancing across multiple pods of a service based on rules or algorithms.
Also handled egress calls for NAT Gateway
ConfigMap
configuration data as key-value pairs.
Store non-sensitive info such as env vars, cli arguments, or config files.
Can be mounted as volumes in pods or accessed as env vars within containers.
Pod Restart not needed after change
Service
Assigned a DNS name and IP allowing other services to discover and communicate with them.
Deployment
Specifications for replicas, update strategies & health checks.
Support Rollbacks, Blue-Green Deployment
Master Node
etcd
K-V Store of Cluster Mgmt Data: Config Data, Resource Metadata, Health Data
Watch API: Other components watch out for change in cluster's state
API Server
It exposes the Kubernetes API, which allows users to interact with the cluster and manage resources.
Central Point of Communication between comp of Control Plane
Handles internal requests from other control plane components for tasks like syncing state or updating configurations.
Maintains required state by managing resource allocation, scheduling pods onto nodes, and monitoring their health status.
Scheduler
Tracks Node: CPU and memory usage, affinity or anti-affinity rules, node capacity, and pod priority to make decisions. Monitors node states and failures.
Assigning pods to nodes based on resource requirements, availability, etc. Optimizes Resource utilisation
Controller Manager
Daemon running in loop.Gathers info & send to API Srvr
You just define the num of pods, services, etc. Ctl Mgr will take care of creating and maintaining that defined config.
Monitors the state of these resources and performs necessary actions to reconcile any deviations.
Also scales up and down according to defined config


K8 Advanced
Stateful Set
For databases, message queues, other stateful workloads.
Redis pod with volume
Cassandra cluster
Webapp using known predefined network ids
Provide guarantees about the ordering, uniqness of pod names.
Gives stable network identity to pods. Post failure, gets replaced with pod with same address
DaemonSet
When need is to run pod(s) on all nodes
Prometheus, New Relic
Monitoring
logstash
Logging
Quobyte, ceph
Cluster Storage
Monitoring
REsource Metrics Pipeline
Gathered by in-memory metrics srvr
Full Metrics Pipeline
Fetches metrics from kubelet and then exposes to K8 via adapter
Cluster Level Metrics
API Req rate, Latency, Errors
Namespace
Grouping Res, sharing res within cluster
Limit Res for each ns, Avoiding conflict, Access control
Security
Role Based Access Control
3rd party auth provider
Isolating from public network
Monitoring network traffic.
Node Failure
Master Node
Worker keeps working. No changes are available/pod creation/svc changes.
Worker Node
If master doesn't receives updates from 5 minutes, marks as NotReady, reschedules all pods to another nodes


K8 Ingress
Load Balancer → Ingress(Reverse Proxy) →K8 Svc Endpoint
Without Ingress
Routing Rules are added as configmap in Nginx
Whenever there is change, configmap gets updated, pods reloaded
Ingress Resource
Comes out-of-the-box
Stores DNS Routing rules
Ingress Controller
Need Nginx/HAProxy separately
Reverse Proxy Srvr deployed as K8 Deployment exposed to Load Balancer
Routes actual req using DNS rules
Workflow
Nginx Pod talks to K8 Ingress API and gets latest routing
For each ingress rule, generate config file and update main config file
When changes are done, config gets updates, graceful reload of config.

upGrad
Building Careers of Tomorrow

Kubernetes Cheat Sheet

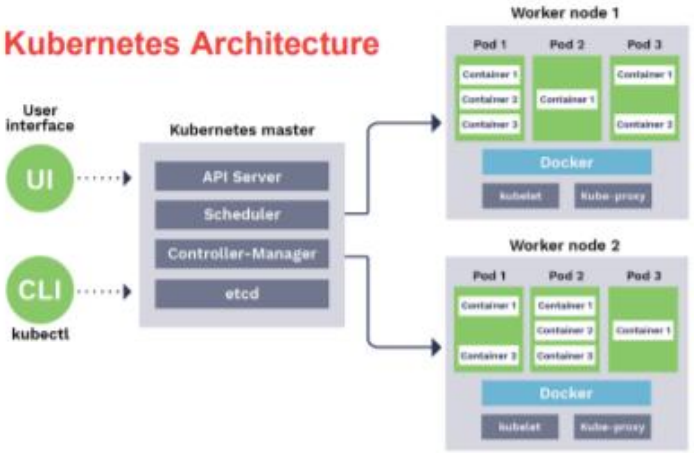
About Kubernetes

Kubernetes is a platform that is designed for managing the life cycle of containerized applications and services completely.

Features

- Maximize resources by making better use of hardware.
- A container orchestrator across multiple hosts.
- Automate the deployment process and updates.
- Able to run a Linux container.
- Auto scaling helps in launching containers on cluster nodes.
- Scaled up and down as per the need.
- Self-healing by replacing, rescheduling, and restarting the dead containers.
- Automated rollbacks and rollouts.
- Load balancing and service discovery.
- Auto restart, auto placement, and auto replication, etc.

Kubernetes Architecture



Components

- API server: Kubernetes API server
- Scheduler: Used for pod scheduling in worker nodes
- Controller: Manages pod replication
- Etcd: A metadata service
- Pod: Group of containers
- Docker: Container based technology, user space of OS
- Kubelet: Container agents that is responsible for maintaining the set of pods
- Kube-proxy: Routes traffic coming into a node from the service

Kubectl Commands

Pods and Container Introspection	
Kubectl describe pod<name>	For describing pod names
Kubectl get pods	For listing all current pods
Kubectl get rc	For listing all replication controllers
Kubectl get rc –namespace="namespace"	For listing replication controllers in a namespace
Kubectl describe rc <name>	For showing the replication controller name
Kubectl get cvc	For listing services
Kubectl describe svc<name>	For showing a service name
Kubectl delete pod<name>	For deleting a pod
Kubectl get nodes -w	For watching nodes continuously

Cluster Introspection	
Kubectl version	For getting version-related information
Kubectl cluster-info	For getting cluster-related information
Kubectl config g view	For getting configuration details
Kubectl describe node<node>	For getting information about a node

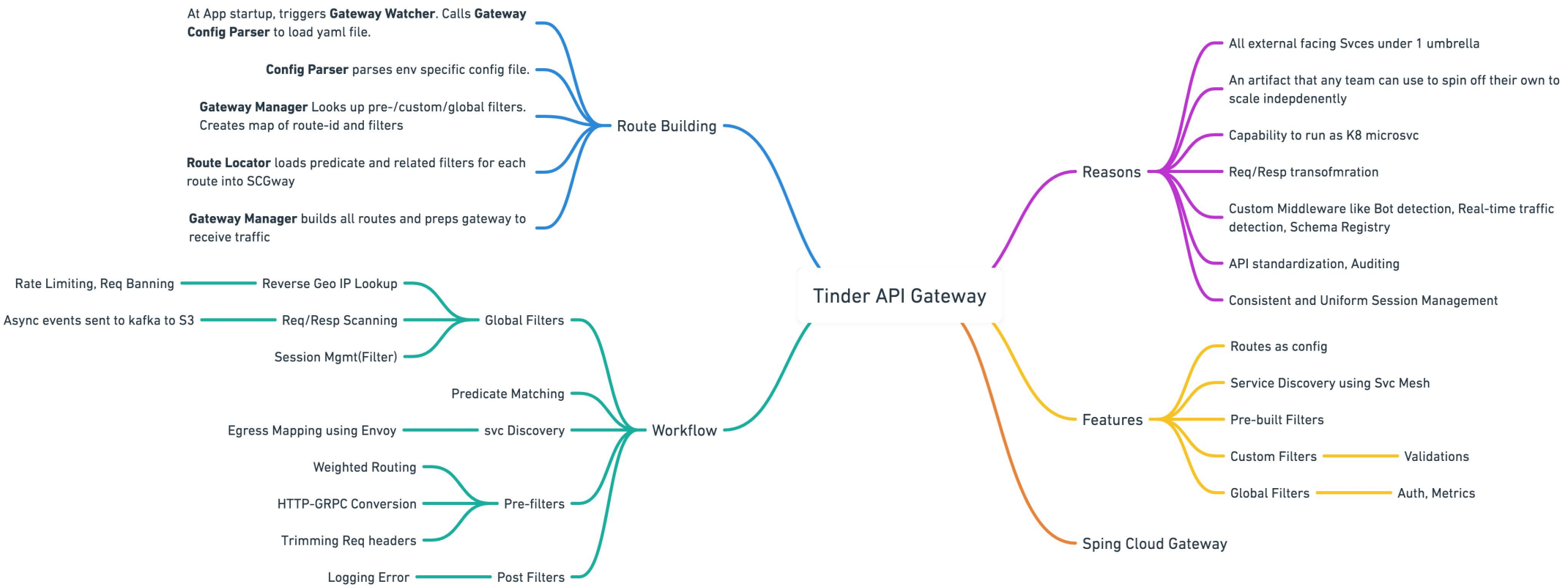
Debugging Commands	
Kubectl top pod	For displaying metrics for a pod
Kubectl top node	For displaying metrics for a node
Watch -n 2 cat/var/log/kublet.log	For watching kubelet logs
Kubectl logs -f<name>>[-c< $container>]	For getting logs from the service for the container
Kubectl exec<service><commands>[-c< $container>]	For execution of the command on service by selecting a container

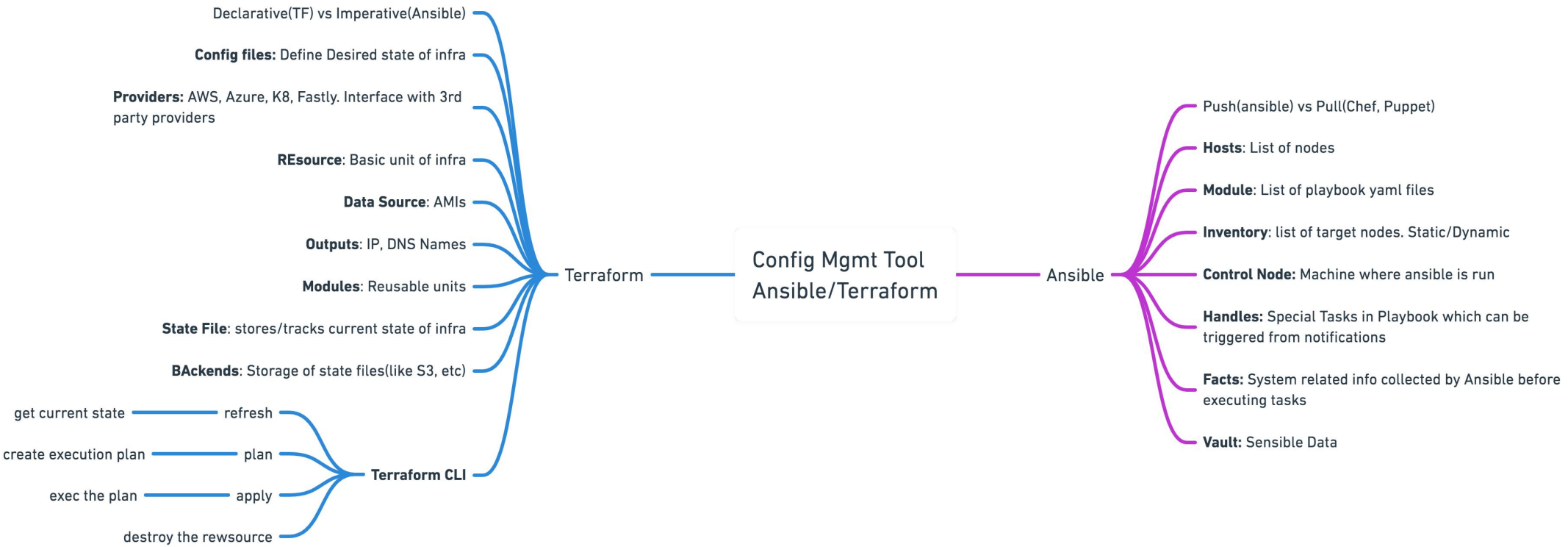
Quick Commands	
Kubectl run<name> — image=<image-name>	For launching a pod with a name and an image
Kubectl create -f <manifest.yaml>	For creating a service described in <manifest.yaml>
Kubectl scale –replicas=<count>rc<name>	For scaling the replication counter to count the number of instances
Expose rc<name> –port=<external>–target-port=<internal>	For mapping the external port to the internal replication port
Kubectl drain<n>– delete-local-data–force–ignore-daemonset	For stopping all pods in <n>
Kubectl create namespace <namespace>	For creating a namespace
Kubectltaintnodes –all-node-role.kuernetes.io/master-	For allowing the master node to run pods

List of Common Objects
All
cm= conf gmaps
Cronjobs
Deploy=deployments
ev= events
jobs
No = nodes
po= pods
Psp= pod security policies
quota= resource quotas
roles
sc= storage classes
clusterroles
crd=custom resource definition
csr= certificate signing requests
ep=end points
ing= ingress
Netpol- network policies
clusterrolebindings
controllerrevisions
cs=component statuses
ds= daemon sets
hpa= horizontal pod autoscaling
limits=limit ranges
ns= namespaces
Pod preset
Pv= persistent volumes
rc= replication controllers
rs= replica sets
secrets
pdb= pod distribution budgets
Pod templates
pvc= persistent volume claims
Role bindings
sa= service accounts
sts= stateful sets

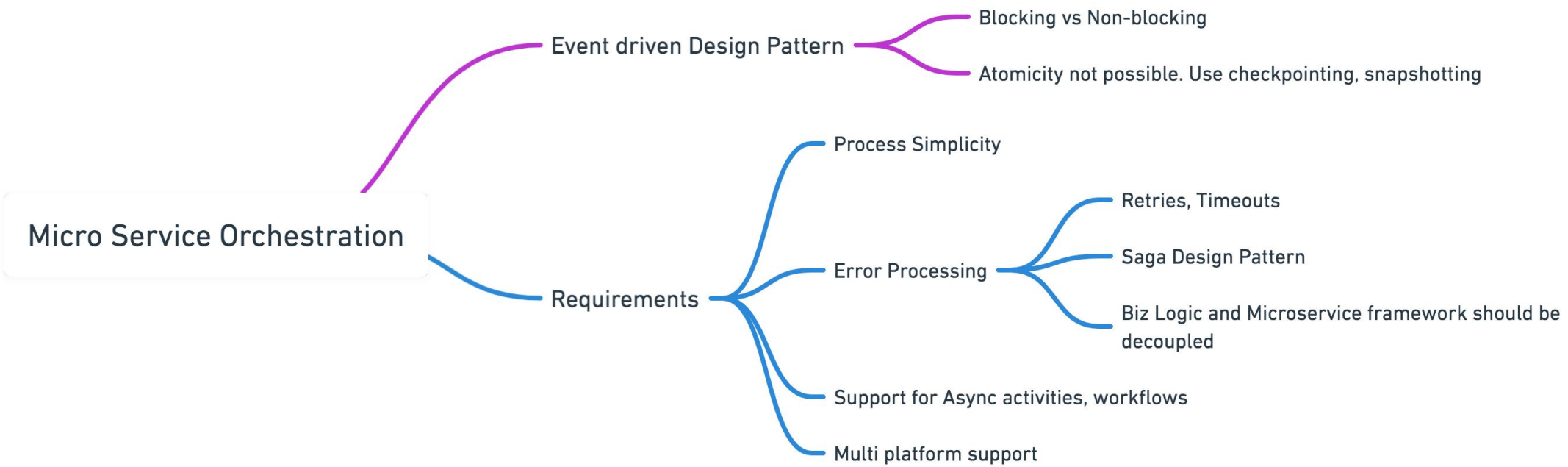
All the basic details of Kubernetes are covered in this., If you are curious to learn more about Kubernetes get in touch with upGrad


Envoy
Features
Dynamic Svc Discovery, TLS Term.
HTTP/2 and GRPC Proxy
Circuit Breaker
Health Check
Staged Rollout with %-based traffic split
Fault Injection
Rich Metrics
Front/Edge Proxy
Main LB to req from Public Internet
TLS Termination
Istio Service Mesh
Usecases
TRaffic Mgmt
Dynamic Svc discovery
Routing
Traffic splitting
Canary release
A/B Testing
Svc-to-Svc Communiction
Reliability(Thru Envoy)
Security
Mutual TLS
Authorization through access policy
Observability
Distributed Tracing
Metrics
Latency, Traffic, Errors, Saturation
Components
Data Plane
Network of Envoy Proxies
Ingress/Egress Traffic
Applications send/receive message to/from localhost
Control Plane
Usecases
Discovery
Cert Mgmt
Config
Istiod
Pilot
COnverts high-level routing rules and traffic control behaviour into envoy specific config
Citadel
Handles Security- Auth,MTLS,Cert
Gallery
Config Validating, Processing and distribution to Pilot
LB aspect
Svc Discovery(available)
Health check
Load Balancing among multiple Srvr
Topologies
Middle Proxy
Centralized
Embedded Client Library
SideCar Proxy
No lock-in language
Sidecar Proxy
Host
Entity capable of network comm
Downstream→Req
Req→Upstream
Listener
Listens on a port for downstream
Similar to Frontend(HAProxy)
Cluster
Group of upstreams
Has its own LB policy
Network Filter
Connect Listener to Cluster
TCP Proxy/HTTP Proxy/Mongo/MySQL Network Filter
Threading Model
Single PRocess multi-threaded model
Each thread is bound to a single connection
No coordination bw threads
Connection Pool
Every Host gets 1 or more connection pool
Connection pools are per worker thread
HTTPS, HTTP/2
Proxying
Layer 7, Layer 4


Tinder API Gateway
Route Building
At App startup, triggers **Gateway Watcher**. Calls **Gateway Config Parser** to load yaml file.
Config Parser parses env specific config file.
Gateway Manager Looks up pre-/custom/global filters. Creates map of route-id and filters
Route Locator loads predicate and related filters for each route into SCGway
Gateway Manager builds all routes and preps gateway to receive traffic
Workflow
Global Filters
Reverse Geo IP Lookup
Rate Limiting, Req Banning
Req/Resp Scanning
Async events sent to kafka to S3
Session Mgmt(Filter)
Predicate Matching
svc Discovery
Egress Mapping using Envoy
Pre-filters
Weighted Routing
HTTP-GRPC Conversion
Trimming Req headers
Post Filters
Logging Error
Reasons
All external facing Svces under 1 umbrella
An artifact that any team can use to spin off their own to scale indepdenently
Capability to run as K8 microsvc
Req/Resp transofmration
Custom Middleware like Bot detection, Real-time traffic detection, Schema Registry
API standardization, Auditing
Consistent and Uniform Session Management
Features
Routes as config
Service Discovery using Svc Mesh
Pre-built Filters
Custom Filters
Validations
Global Filters
Auth, Metrics
Sping Cloud Gateway


Config Mgmt Tool
Ansible/Terraform
Terraform
Declarative(TF) vs Imperative(Ansible)
Config files: Define Desired state of infra
Providers: AWS, Azure, K8, Fastly. Interface with 3rd party providers
REsource: Basic unit of infra
Data Source: AMIs
Outputs: IP, DNS Names
Modules: Reusable units
State File: stores/tracks current state of infra
BAckends: Storage of state files(like S3, etc)
Terraform CLI
refresh
get current state
plan
create execution plan
apply
exec the plan
destroy the rewsource
Ansible
Push(ansible) vs Pull(Chef, Puppet)
Hosts: List of nodes
Module: List of playbook yaml files
Inventory: list of target nodes. Static/Dynamic
Control Node: Machine where ansible is run
Handles: Special Tasks in Playbook which can be triggered from notifications
Facts: System related info collected by Ansible before executing tasks
Vault: Sensible Data


Authentication
HTTPS
TLS
Transport Layer Security
Steps
TCP Handshake
TCP Sync
TCP Sync, Ack
TCP Ack
Certificate Check
Client Hello
informs TLS Version
informs Cipher Spec(Eg. RSA)
Server Hello
Srvr→Certificate
Public Key
Key Exchange
Client creates session key. Depending on Cipher Spec, encrypts session key using public Srvr Key. Sends to Srvr.
Srvr decrypts it using public key.
Data Transmission
Encrypt data using Session Key. Exchanges encrypted data
OAuth2
Token
Auth code, access token, refresh token.
Srvrs
Identity Provider, Authorization Srvr, Resource Srvr
Flow
User Reqs resources using client app
Client app gives client-id, scope to initiate Auth Provider Login Flow
User logins into Request Perm. Dialog by auth Provider
Auth Provider redirects to URI and authentication code.
User access the URI→Client App gets auth code
Client app sends auth code, client-id, client-secret to Auth Provider to get access token
CLient app uses access token to get resources from Resource Srvr.
Client App uses refresh token to acquire new access token
JWT
Header
Algo
HMAC SHA 256, RSA, etc
Token Type
JWT
Payload
Actual data base64url encoded
Signature Segment
BAse64 Url encoded Header and Payload, Secret Key. Hashed using algo defined in header
Client can use header,payload with private key to verify the signature.


Micro Service Orchestration
Event driven Design Pattern
Blocking vs Non-blocking
Atomicity not possible. Use checkpointing, snapshotting
Requirements
Process Simplicity
Error Processing
Retries, Timeouts
Saga Design Pattern
Biz Logic and Microservice framework should be decoupled
Support for Async activities, workflows
Multi platform support