Nomad Cheatsheet

A quick reference guide for HashiCorp Nomad, covering essential commands, concepts, and configurations for job scheduling and cluster management.

Nomad Basics

Core Concepts

Client: Executes tasks on behalf of Nomad.
Server: Manages the cluster state, schedules jobs, and handles client communication.
Job: A declaration of tasks to be run and their requirements.
Task: A single unit of work within a job.
Allocation: A mapping of a task to a specific client.

Driver: Responsible for executing tasks. Examples include `docker`, `java`, `exec`, `raw_exec`.

Nomad CLI Commands

`nomad job run <jobfile.nomad>`	Submit a job to Nomad.
`nomad job status <job_id>`	Check the status of a job.
`nomad job stop <job_id>`	Stop a running job.
`nomad node status`	Show status of all the nodes.
`nomad alloc status <alloc_id>`	Show status of the allocation
`nomad status`	Displays the overall Nomad cluster status.

Basic Job File Structure

```
job "example" {
  datacenters = ["dc1"]
  type = "service"

group "web" {
    count = 3

task "server" {
      driver = "docker"

      config {
        image = "nginx:latest"
        port_map {
          http = 80
        }
      }

      resources {
        cpu    = 500
        memory = 256
        network {
          mbits = 10
          port "http" {}
        }
      }
    }
  }
}
```

Job Specification Details

Job Block

`job "job_name" {}`	Defines the job. Must be unique within the datacenter.
`datacenters = ["dc1"]`	Specifies the datacenters where the job can run.
`type = "service"`	Job type. Can be `service` (long-running) or `batch` (finite).
`priority = 50`	Specifies job priority. Higher number means higher priority. Default is 50.
`update {}`	Controls the job update strategy.

Group Block

`group "group_name" {}`	Groups tasks together for scaling and placement.
`count = 3`	Number of task instances to run in this group.
`restart {}`	Defines restart policy for tasks in the group.
`ephemeral_disk {}`	Configures an ephemeral disk for tasks in the group.
`constraint {}`	Defines constraints for task placement.

Task Block

`task "task_name" {}`	Defines a single unit of work to be executed.
`driver = "docker"`	Specifies the task driver to use (e.g., docker, exec).
`config {}`	Driver-specific configuration (e.g., Docker image, command).
`resources {}`	Specifies resource requirements (CPU, memory, network).
`service {}`	Defines how the task should be registered as a service.
`template {}`	Configures dynamic templates using Consul or Vault data.

Advanced Features

Constraints

Constraints ensure that tasks are placed on suitable clients based on attributes.

Example:

```
constraint {
  attribute = "${node.class}"
  operator  = "=="
  value     = "web"
}
```

Common attributes: `node.class`, `node.datacenter`, `driver.docker`.

Update Strategy

`update {}`	Controls how jobs are updated (rolling updates, canary deployments).
`max_parallel = 1`	Maximum number of allocations that can be updated concurrently.
`stagger = "10s"`	Delay between updating allocations.
`min_healthy_time = "30s"`	Minimum time an allocation must be healthy before continuing.
`auto_revert = true`	Automatically revert to the previous version if the update fails.

Templates

Templates allow dynamic configuration based on Consul or Vault data.

Example:

```
template {
  data = <<EOH
  {{ with secret "secret/data/mydb" }}
  DATABASE_PASSWORD={{ .Data.password }}
  {{ end }}
  EOH

destination = "secrets.env"
  perms = "0644"
}
```

Networking and Service Discovery

Networking

`network {}`	Configures the network resources for a task.
`port "http" { static = 8080 }`	Defines a static port mapping.
`port "http" {}`	Defines a dynamic port mapping, assigned by Nomad.
`mbits = 10`	Configures network bandwidth in megabits per second.

Service Discovery with Consul

Nomad integrates with Consul for service discovery.

Example:

```
service {
  name = "web"
  tags = ["v1"]
  port = "http"

  check {
    type     = "http"
    path     = "/health"
    interval = "10s"
    timeout  = "5s"
  }
}
```

This registers the task with Consul, including health checks.

Vault Integration

Nomad can retrieve secrets from Vault for secure configuration.

Example:

```
template {
  data = <<EOH
  {{ with secret "secret/data/mydb" }}
  DATABASE_PASSWORD={{ .Data.password }}
  {{ end }}
  EOH

destination = "secrets.env"
  perms = "0644"
}
```

Ensure that the Nomad client has appropriate Vault policies.