- Implementation
    - Phase 1 basic command
        - “install”: pull the nomad package from a url, and “nomad run” behind the scene (docker images could also be applied here)
        - “uninstall”: “nomad stop” the package behind the scene, and provide option to delete downloaded files
        - “source”:
        - “search”: search if a particular package exist or not
    - reference application: wordpress redis sql

    - Phase 2
        - provide persistence storage volume for running tasks (connect to a local host database, CSI plugins)
        - provide inter-communication for running tasks
    - Phase 3
        - Nomad registry(unofficial website)
- Features that may be needed
    - Do not directly edit the nomad file, just use the command line to complete a series of changes in nomad file -> to avoid configure wrongly
- Deploying WordPress(a load-balanced web application) on Kubernetes
    - https://www.youtube.com/watch?v=-Hn7vFAr-9s
    - Tech: Kubernetes, digitalocean(service). create separate mysql and wordpress pods, each with their own data volumes
    - Structure:
        - Some config data (k8s ConfigMaps and Secrets)
        -
        - MySQL Container (k8s replicaset)
        - MySQL Service (k8s service)
        - |
        - WordPress Container (k8s deployment)
        - [apache and php-fpm]
        - |
        - DO Loadbalancer (k8s service)
    - Question: which service/database
- Prior art for Nomad job reuse
    - levant
        - what
            - open source templating and deployment tool for HashiCorp Nomad jobs that provides real time feedback and detailed failure messages upon deployment issues
            - thinking:
                - How to define which information is necessary for successful building?
                - how to define or analyze Failure?
                - Why only Canary Auto Promotion? How about other Deployment Patterns?
        - Workflow

            - git clone https://github.com/hashicorp/levant.git the repo on your local dir
            - make build in local dir
            - the executable file is located in `./bin/`
    - Backpack
        - packaging system (like Helm on Kubernetes)
        - what it does
            - Help define and install complex jobs configuration
            - Helps building reproducible jobs across multiple Nomad clusters
            - Simplifies updates to new version of jobs
            - Allows publish and share packages of applications
        - Detailed tool description: https://blog.setale.me/2020/11/12/backpack-helm-charts-but-for-hashicorp-nomad/
            - a great template for us to learn if we need to recommend our product to others
                - why need this
                - how to use it
                - future plan
        - Workflow
            - Prerequisite: run a nomad agent beforehand
                - sudo nomad agent -dev
            - **Create** your first pack, by using the boilerplate directory structure:
                - backpack create nginx
            - **Pack** all the files into one single pack:
                - backpack pack ./nginx-0.1.0/
            - **Customize** the values for the template to configure, enable, adjust the jobs:
                - backpack unpack values ./nginx-0.1.0.backpack -f ./values.yaml
            - **Plan** and validate (dry-run) the jobs of a package before running:
                - backpack plan ./nginx-0.1.0.backpack -v ./values.yaml
            - **Run** your Nomad Jobs with my custom values:
                - backpack run ./nginx-0.1.0.backpack -v ./values.yaml
            - **Check** the status of the job allocations:
                - backpack status ./nginx-0.1.0.backpack --all
            - **Unpack**, **customize** or **Run** a backpack **from an URL**:
                - backpack unpack values https://backpack.qm64.tech/examples/redis-6.0.0.backpack -f ./values.yaml
                - backpack run https://backpack.qm64.tech/examples/redis-6.0.0.backpack -v values.yaml

- Reference APP

- wordpress
  - what - demonstrates several useful patterns for creating Nomad jobs:
    - Nomad Host Volumes for persistent storage
    - Using a pre start task to wait until a dependency is available
    - Template driven configuration to minimize static port references
  - prerequisite - Consul
    - locate the supporting MySQL instance
  - **workflow**
    - Add the host_volume information to the client stanza in the Nomad configuration
      - sudo touch /etc/nomad.d/client.hcl (config file)
      - in client.hcl

        client {
        enabled = true
        host_volume "my-website-db" {
        path = "/opt/volumes/my-website-db"
        read_only = false
        }
        }
    - Acquire wordpress.nomad file (nomad job file)
    - Create a folder on one of the Nomad clients to host the registry files
      - sudo mkdir /opt/volumes/my-website-db
    - Establish nomad client
      - Start nomad agent with -config to load config file (start agent)

        sudo nomad agent -config=/etc/nomad.d/client.hcl -dev -bind 0.0.0.0 -log-level INFO
    - Restart Nomad to read the new configuration
      - Restart Nomad service to load the config file
      - run wordpress.nomad (run job file)

        inside wordpress.nomad dir:

        nomad job run wordpress.nomad
    - Reference Link:
      - https://learn.hashicorp.com/tutorials/nomad/production-deployment-guide-vm-with-consul#client-configuration
      - https://github.com/angrycub/nomad_example_jobs/tree/master/applications/wordpress
      - https://learn.hashicorp.com/tutorials/nomad/get-started-run?in=nomad/get-started
- minio

        - what - uses Nomad Host Volumes to provide an internal s3 compatible storage environment which can be used to host private artifacts for a Nomad clusters.
        - prerequisite - Consul
            - use to locate the MinIO instance
        - workflow
            - Establish nomad client and consul client
            - Create a folder on one of the Nomad clients to host the registry files
            - Add the host_volume information to the client stanza in the Nomad configuration
            - Restart Nomad to read the new configuration
- Tools from Nomad
    - Consul
        - a service networking solution to automate network configurations, discover services, and enable secure connectivity across any cloud or runtime
    - Terraform
        - a tool for building, changing, and versioning infrastructure safely and efficiently

- Related tool
    - **terraform registry**
        - gives **Terraform** users easy access to templates for setting up and running their infrastructure with verified and community **modules**
        - workflow
            - search required module from Terraform registry official website https://registry.terraform.io/
            - locates the module; Copy and paste into Terraform configuration, insert the variables, and run `terraform init`
    - **Helm Chart**
        - package management tool for Kubernetes.
        - Helm charts provide templating syntax for Kubernetes YAML manifest documents. Configurab
        - le deployments instead of static files.
        - chart: packaging format for Helm. A collection of files that describe a related set of Kubernetes resources.

    - **Advanced Package Tool (Ubuntu)**
        - a software user interface for managing software on Unix-like computer systems.
        - automates the retrieval, configuration and installation of software packages, either from precompiled files or by compiling source code.
        - apt: provides a high-level command line interface for the package management system. Intends as an end user interface and enables some options better suited for interactive usage by default.
    - **NPM**
        - JavaScript Package Registry

- **Homebrew**
  - By default, Homebrew can only install core Homebrew Formulae**.**
    - To install a third-party package, need to run brew tap first.
  - Brew install
    - Steps:
      - fetch the URL using **curl** and stores the archive in a cache directory (avoid duplicate download)
      - computes **checksum** validate the downloaded file authenticity
      - execute the **install** method defined in the formulae6
    - When run brew install, Homebrew reads the package's Formula — an implementation of Ruby's abstract Formula class that **provides package metadata and installation instructions** — create an executable from our source code and install it locally on our computer.
    - Homebrew is following our Formula by: downloading and installing our dependency, downloading our source code, and running cargo build(Rust) — release — bin hello to compile our code and put our binary in /usr/local/bin.
    - Homebrew installs packages into /usr/local/bin because this directory location is already on your Mac's PATH — this means you can execute the newly created binary by name without providing an explicit path to it.

Questions