Table of contents

Created by Igor for Plumsail March 2024
Design of the project based on the test task

Test task for Sysadmin/DevOps

Our globally used **web service** consists of **web applications** and **jobs** and uses Postgres, RabbitMQ, and Redis. The web applications run on **Linux** while the jobs work on **Windows**.

We want to have isolated environments for serving customers in different regions: the US, EU, and Australia. Each environment must work **independently**. For instance, we can release a new feature for US-based customers only and later, publish it for customers in other regions. While we have a good estimate of regional customer numbers, our environments must be designed for easy **scalability**. Additionally, we must have the flexibility to further **subdivide regions** in the future, such as splitting the US environment into US-east and US-west for reducing latency for US-based customers. Access to these environments must be restricted to **administrators** only.

Also, we need shared infrastructure for **building**, **testing**, **deploying**, **logging**, and **monitoring** activities with access by both groups, **developers and administrators**.

We can rent any number of hardware servers (32 GB RAM, 512 TB SSD) in any region. Each has one white IP address and 100MBps bandwidth.

With the above requirements and limitations, how would you organize:

- Overall architecture
- Scalability within one region
- Provisioning of a new region
- Virtualization system
- CI/CD automation
- Security control

Analysis of Constraints for Design.

Based on the task, the use of specialized network devices for load balancing and fault tolerance, as well as NAS DAS storage devices, is not implied. There is no mention in the task of using cloud services such as Azure AzureAD, AWS, Cloudflare, and others.

It is obvious that the main limiting factor will be the network bandwidth of 100 megabits and the presence of only one NIC per server. This prevents us from creating full-fledged clustered assemblies for evenly distributing the load at the hypervisor level or at the level of database HA or replicas, which require dedicated NICs for inter-server communication, heartbeat, and direct access.

Container clusters based on k8s k3s are not applicable for the same reason.
A 100-megabit connection is approximately 35 gigabytes of data per hour under ideal conditions, which gives us the opportunity to try to implement the Database Replica approach in one region, but the design below will be written based on the fact that our databases are completely independent.
Kubernetes in our case will only be used for making the CI/CD process, not for load balancing or resource management.

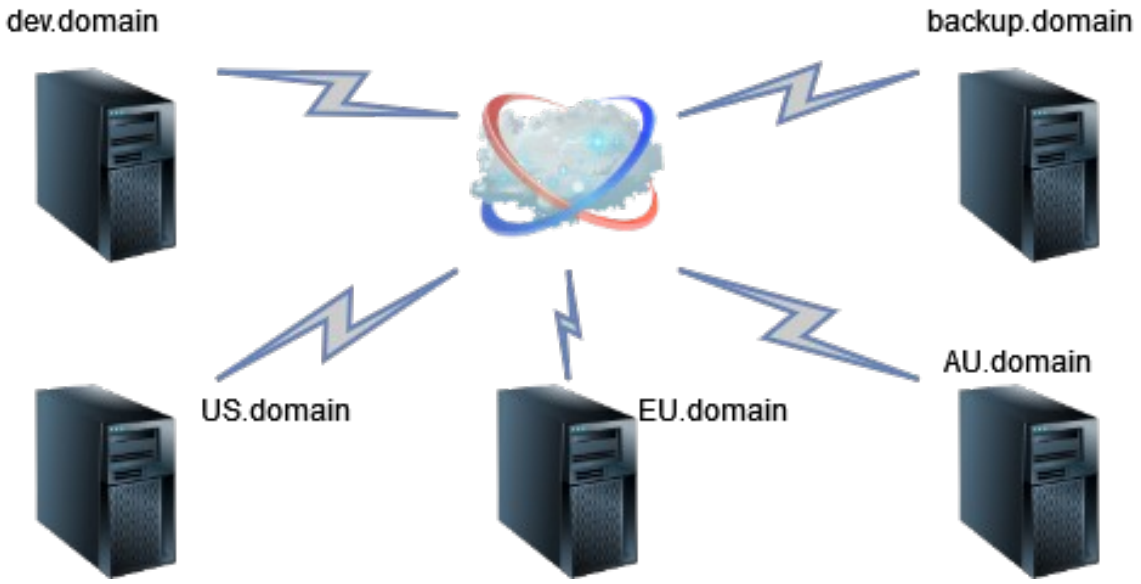
The second limiting factor will be the size of the storage system at 512 gigabytes per server, which cannot be expanded in case of exhaustion. (512**TB** ssd must be typo at task discription ?)

Conclusion: each server must be an entirely independent node containing all the necessary components for the application to function.

Overall architecture

Each server should be a fully independent node containing all the necessary components for the application to function.
The main domain **"domain"** is served by all servers as the primary site, allowing for load distribution across regions and ensuring site availability and fault tolerance.
Each region represents an independent server with a subdomain of the form XX.domain, accessible both via the main **"domain"** address and the specific address of this subdomain «**XX.domain»**.
See the DNS section for more details.

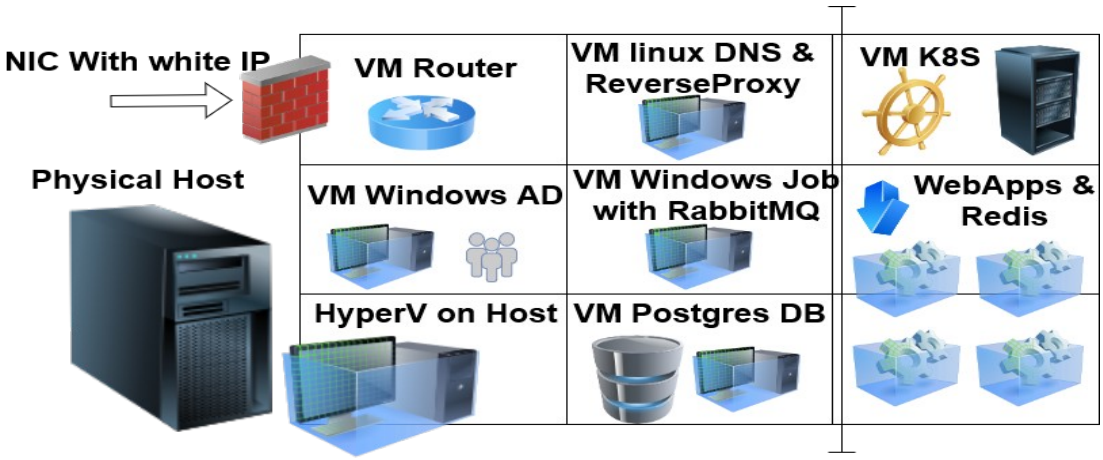
Additionally, we use a server **Dev** for development purposes and a server for storing database backups. The development server could potentially be used as a DR server.



All nodes are connected via **Wireguard**, With a point-to-point mesh connection configuration, ensuring server availability within a single virtual network.

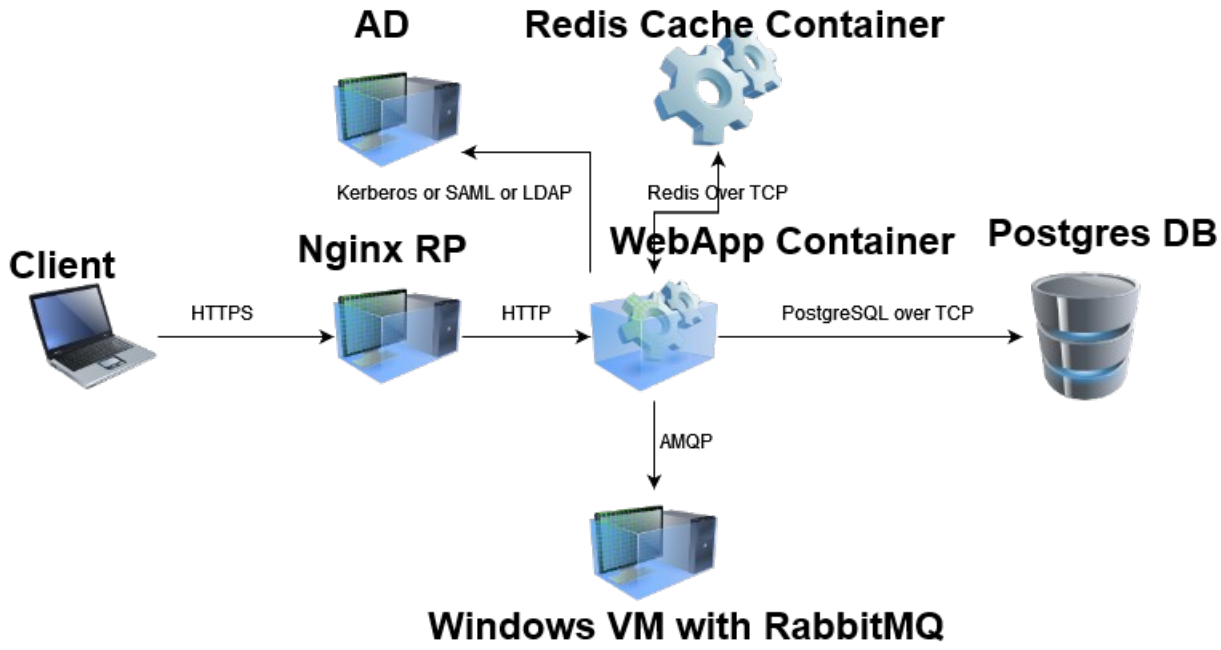
Each server is equipped with a set of components:

- A hardware server is equipped with the **HyperV** hypervisor (or Windows Server with the HyperV role),
- **Wireguard** (as a virtual router or application on the host itself),
- **Minikube** as the main container orchestrator for the application,
- A virtual machines with the **PostgreSQL** database, **Windows AD DS**, **Windows job** server, **RP Nginx**



Some roles and applications, such as PostgreSQL (instead postgres containers), Windows VMs are intentionally deployed outside of Minikube to reduce network and IO overhead costs.

Application work logic on each node will be like :



Scalability within one region

In its current form, the design does not imply an elegant option for scalability within one region, as the database is unique within the boundaries of each node. Therefore, to enable scalability within one region, it is proposed to use the same approach as Provisioning of a new region.

The subdivision of regions required in the task can be achieved by:

1) Provisioning a new server like for a new region.
2) Cloning the DB VM or restoring the DB of the region from a backup, restoring only records related to users in a new region.
3) Removing users related to a new region from the old DB.

However, within the scope of the task, it is worth mentioning the theoretical possibility of database replication within one region if it aligns with the business logic of the application, database size, update speed, and other factors.

Replicating to Australia definitely does not make sense due to huge delays; the AU region is proposed to be created exclusively autonomously in any scenarios.

That is, two or more servers could serve US-east and US-west simultaneously, having a consistent set of data in the replicated database.

The speed of operation would be ensured by caching data in Redis, and delays in processing jobs on the Windows server and the web app in containers would be minimized.

When placing a group of servers in one data center with inter-server connections, it would allow us to consider options for High Availability, HAProxy, load distribution using a Kubernetes cluster, adding all Windows servers to a RabbitMQ queue for load distribution, and other HA approaches. However, this is a completely different level of budget and complexity for the project.

Provisioning of a New Region

The deployment of a new region will involve:

1. Ordering a new certificate for the Reverse Proxy, updating the configuration of other RPs with the new subdomain.
2. Updating the DNS zone with a record for the new region.
3. Deploying a standard server with a hypervisor and basic VMs.
4. Updating the network configuration, namely:
    - Adding the IP address of the new server to the white lists of firewalls on the old servers.
    - Updating the Wireguard configuration and routes to the new subnet on the old servers.
    - Adding the server to the monitoring system.
5. Replicating container images with the application and functions for this region and applying the YAML scenario to Minikube for production launch.
6. Updating documentation, passwords, and access keys to the server for other team members' access.

It is proposed to automate these actions using Ansible. Preparing playbooks for standard actions in advance would facilitate this automation.

Virtualization System

Choice and Justification of Hypervisor

Since the design requires us to use Windows, one Windows Server Standard license covers the use of Hyper-V on a bare-metal platform with an additional two virtual Windows servers.
Additionally, we will leverage the benefits of Active Directory for authentication and storage of user accounts, both for company employees and application users.

Of course, it is possible to develop a design based on other popular hypervisors such as XCP-NG, KVM, or Proxmox.
However, the use of commercial solutions like VMware and Xen is likely to be unjustified.

As for the container orchestrator, it is proposed to use Kubernetes implemented in Minikube with the Hyper-V virtualization subsystem.

CI/CD automation

Infrastructure Setup
1. Minikube will be utilized on the DEV server with full access granted to the Application team.
2. An account on Docker Hub will be set up if not already done.
3. A new project will be created on Docker Hub for the application.

Setting up Git Repository
1. A Git repository will be established for the application.
2. The Git repository will be linked with Minikube on the DEV server to enable automatic builds upon code changes.

CI/CD Pipeline Configuration
Creating Configuration File for CI/CD
1. Create a CI/CD configuration YAML file.
2. Configure this file to execute the following steps:
   - Pull source code from the Git repository.
   - Build Docker image of the application.
   - Test Docker image (optional).
   - Push Docker image to Docker Hub.
   - Update Kubernetes (using Minikube) with the new Docker image.

Testing the CI/CD Pipeline
1. Push the changes to the Git repository.
2. Monitor the build and deployment process in the CI/CD tool.
3. Ensure all steps are successfully executed.

Application Deployment
Upon successful completion of the CI/CD pipeline, the application will be automatically deployed using the new Docker image version on Dev Minikube.

Application Deployment to Production Infrastructure
The Administrator team, with appropriate RBAC access permissions, will deliver the updated new version of the Docker image to agreed regions nodes using DFS-R replication share. Using Deployments YAML replace pods on a new app version under control.

Monitoring and Maintenance
1. Set up monitoring for the application using Kubernetes monitoring tools (e.g., Prometheus and Grafana).
2. Verify that the monitoring is functioning correctly and receive notifications for any issues.

Security Controls

1) Firewall with a whitelist is configured on all hosts to allow only the IP addresses of our servers from other regions.
Ports for DNS, HTTP, HTTPS, and Wireguard (custom) are permitted, while all other protocols are denied.

Consideration can be given to implementing fail2ban systems and limiting the number of incoming HTTP requests from a single host using NGINX.

RBAC is applied both in the Windows domain and on Kubernetes hosts.
NetworkPolicy s applied both in the Windows domain and on Kubernetes hosts (need to ajust CNI plugin, for example on calico cni).

2) Only personalized accounts are used.

For RDP access to the dev server, 2FA for RDP sessions can be implemented.

A heightened level of auditing is applied to user actions on the Dev server.

3) All work is conducted exclusively on the Dev server, accessed via Wireguard.

Access to the server can be organized via SSH or working through RDP on the Windows machine of this server.

4) In monitoring, triggers are set for logons on all servers for any remote management protocols (SSH, RDP).

If deploying a virtual machine with Kubernetes on the Dev server, regenerate SSH keys before moving to the new node.

5) The administration team regularly monitors the release of CVEs and security bulletins for the operating systems and devices used.
In the event of such publications, patches or workarounds are applied.

The impact and maintenance downtime for the service are coordinated with the business according to the risks posed by vulnerabilities.

Additionally, regular patching of all systems is conducted according to a schedule.

DNS

For the project, the domain name "domain" is registered, and a certificate for HTTPS encryption is purchased.

Each server will receive an independent subdomain according to its location:
- US.domain
- EU.domain
- AU.domain
- Dev.domain

In the future:
- USwest.domain
- USEast.domain

GeoDNS

We will manage DNS zones independently, enabling us to implement GeoDNS technology for proper user distribution according to their region. This approach also allows for flexible addition and removal of nodes and regions from usage.

Each server will have a virtual machine with Bind9 servicing the main domain domain and its subdomains. The configuration will be identical on all servers.
When accessing the shared zone, GeoDNS will activate, and the client will be returned the IP address of the nearest node.

It's important to note that in the event of a failure or maintenance of one of the servers, manual DNS settings adjustment is required to prevent routing to the inaccessible node. During this scenario, users in that region will be unable to use the application, but the main domain "domain" served by other nodes will remain accessible, partially preserving service provision (the ability for users to use the service in other regions).

However, potential issues may arise if users are utilizing VPNs, proxies, or third-party DNS providers.

A possible solution could involve implementing business logic on the website, such as choosing the correct currency or language, which would redirect users to the corresponding subdomain. Alternatively, user authentication could determine the user's location and redirect them to the node where their data is stored. The use of cookies, reading reverse proxy cookies, and redirecting users to the node where their data is located could also be considered.

Backups / DR Plan

Backups of databases from all nodes are proposed to be stored on a dedicated server. Database-level restoration would allow for user migration from region to region.

The disaster recovery plan involves:
1) Excluding the failed node from DNS on all nodes and replacing it with the IP of the Dev server.
2) Editing routes and Wireguard configuration for all nodes to exclude the failed node and add a subnet route of this node to the internal IP of the Dev server.
3) Editing the NGINX RP configuration on all nodes to redirect the zone to the Dev server.
4) Restoring the database on the Dev server from backup.
5) Launching the web application in Kubernetes with functionality relevant to the lost node.

It is suggested to write Ansible playbooks for different scenarios in advance.

Patching Plan / Server Maintenance

Patches are initially applied to the Dev server.

If no issues arise during this process, they are gradually applied to the rest of the nodes by region. During the application of updates, the functionalities of the application on the node's region will be unavailable, but the main site will remain accessible.

It is possible to automate the exclusion of nodes from DNS zones during patching.

Monitoring

It is suggested to use monitoring tools such as Zabbix with a server on the Dev node and a backup server on the backup server. The main monitoring triggers include:
- Server availability
- Percentage of available resources (memory, disk, disk latency)
- Percentage utilization of the host NIC
- Logons to any production servers
- Scenarios for continuous testing of HTTP and DNS responses from all nodes.

Network

A virtual router, such as PFsense, Microtik, OpenWRT, or a Linux VM with IPv4 forwarding and configured iptables, is essential.

This component is not only necessary for maintaining Wireguard connections but also for implementing firewall, QoS, and queue limits on connections or protocols (to prevent DoS/DDoS attacks).

It's crucial to set limits in a way that allows the server to maintain minimal accessibility for administration during channel overload.

Each subdomain is serviced by a virtual subnet of the form 192.168.x.0/24 for hosts in the subdomain, including HyperV hosts, AD DS, Minikube, NGINX Reverse Proxy, and Windows Job. This ensures uniform addressing across all nodes while providing subnet-level isolation.

For example:

Region	White IP	Wireguard	Network	HypervHost	AD Windows	BackupDB
US	1.1.1.1	10.0.0.1/32	192.168.1.0/24	192.168.1.2	192.168.1.3	-
EU	2.2.2.2	10.0.0.2/32	192.168.2.0/24	192.168.2.2	192.168.2.3	-
AU	3.3.3.3	10.0.0.3/32	192.168.3.0/24	192.168.3.2	192.168.3.3	-
Dev	4.4.4.4	10.0.0.4/32	192.168.4.0/24	192.168.4.2	192.168.4.3	-
Backup	5.5.5.5	10.0.0.5/32	192.168.5.0/24	-	-	192.168.5.55

The transport subnet for Wireguard is 10.0.0.1/24, but in practice, these will be point-to-point connections between each server. Reverse routes for each subnet of other regions are configured for connectivity.

It is proposed to update routes and Wireguard configuration via Ansible, preparing playbook templates in advance for scenarios like:
1) deploying a new region,
2) removing a region,
3) modifying a region.

Setting up HyperV Host

Main Approach:
For network interaction, the physical NIC of the HyperV host is protected by the standard Windows firewall, and necessary ports (TCP 80, 443, 53) are forwarded to the virtual network card of the HyperV virtual switch to the IP address of Minikube.
Wireguard is installed directly on the host or on a Linux VM.

Alternative Approach:
This approach offers greater network performance and security as it bypasses the HyperV host. However, an alternative method of connecting to the server should be available (e.g., IPMI, IRMC, iLO, iKVM).

For network interaction, the physical NIC of the host is bound to a virtual machine performing router and firewall functions. This VM can be a virtual PFsense, Microtik, OpenWRT, or a VM with NGINX Reverse Proxy installed, along with Wireguard, IPv4 forwarding enabled, and iptables configured.

Important Note: Loss or misconfiguration of this VM will result in loss of access to the entire server.

The design was compiled by Igor for Plumsail March 2024