**Rationale**

**Problem:**

* The client is currently hosting 10 WordPress sites using WordPress Multisite in a private datacenter. *Multisite is a WP feature which allows users to create a network of sites on a single WP installation.* So in the figure wp sites 1 through 10 are subdomains or subfolders of the main wp app.
* They achieve high availability (HA) by using 2 servers and having two copies of their Multisite. For the database, they are using 2 MySQL servers behind and HAProxy to achieve HA. *The HAProxy that acts as the load balancer allows the web server to spread incoming requests across multiple endpoints.*
* The past few months, they have been having a lot of issues because some of their websites have increased in popularity, especially during certain timeframes. *If one website needs more resources, it affects all the other websites hosted in the same server.*
* For the future state, they have agreed that they want to move away from Multisite, and have independent WordPress applications. They have also pointed out that they have 5 more sites in the making that will reach Production in the next 12 months. *Knowing that they’re already experiencing performance issues with the current setup, there’s a big possibility that they will encounter more issues if the additional sites go live.*

**The solution needs to be:**

* scalable and flexible *- this means that the system should be able to handle increased load and the system can adapt or respond to internal and external changes*
* future proof and expandable with new WordPress sites with minimal effort *- the system should not be significantly updated as technology changes. Another thing is that we should be able to deploy new sites into production smoothly without affecting the site's performance.*

**Proposed Solution:**

* The microservices architecture style will be used for designing the overall architecture
* The Azure Kubernetes Service (AKS) will be used as the compute service
* The Azure Database for MySQL is the managed, relational database service used
* Each website/application will be represented as a service / deployment on the Kubernetes cluster
* The keys and other secrets are stored in Azure Key Vault
* The code repository is a multi-repo, with folders organized by microservice
* The team’s branching strategy is based on trunk-based development
* The team uses release branches to manage releases. Separate releases are created for each microservice.
* The CI/CD process uses Azure Pipelines to build, test and deploy the microservices to AKS
* The container images for each microservice are stored in Azure Container Registry
* The team uses Helm charts to package each microservices

***Considerations:***

The Azure Application Architecture Guide was used for the design process. This guide presents a structured approach for designing app on Azure that are scalable, resilient and highly available.

**Architecture Styles** - The Microservices architecture was used for the design. Here, each service implements a single business capability and can be deployed without a lot of coordination between the teams which encourages frequent updates. By using microservices, we can build an application as a suite of small services running in its own process and are independently deployable. We can easily isolate defects on the system too. If the developer make some changes on a service, it won't affect the other services. There's no need to restart the whole system if a service fails. Since they have 10 websites, if they want to make changes on one website, we can still guarantee that it won't affect the performance of the other sites.

Even though microservices architecture is more complex to build, if done right, this style can lead to higher release velocity, faster innovation and a more resilient architecture. We can create a system where services can be deployed independently, faults are isolated, frequent updates are possible and it's easy to introduce new technologies into the application.

**Technology Choices**  - This is based on the compute service and data stores.

*Compute service*. I used the Azure Kubernetes Service (AKS) because it manages a hosted Kubernetes service for running containerized application. AKS is a free service so there's no charge on the cluster management. You only pay for the VM's, associated storage and networking resources consumed. Since I mentioned earlier that I'll be using microservices architecture, AKS is the appropriate compute service for that. AKS automatically configures all K8s masters and nodes during deployment process, and handles a range of other tasks including Az AD integration, connection to monitoring services and configuring advanced networking features, such as HTTP application routing. AKS nodes can scale up or down to accommodate fluctuation in resource demands. Microsoft handles all K8s upgrades for the services, as new versions become available. Users can decide when to upgrade the K8s version in their own K8s cluster to reduce the possibility of accidental disruption.

The *data store* is using the Azure Database for MySQL. This provides a fully managed, enterprise-ready community MySQL DB as a service. It helps lift and shift to the cloud using different language and framework of choice. It has built-in high availability and dynamic scaling, helping you easily adjust to changes in customer demands. It has unparalleled security and compliance, including Azure IP advantages, as well as Azure's industry leading reach. And it has flexible pricing model with no hidden cost.

The following were the considerations based on the quality pillars used when designing the Architectural Design of the WordPress application:

**Scalability**. Kubernetes supports scaling the number of pods allocated to a deployment and scaling the nodes in the cluster to increase the total compute resources available to the cluster. Auto Scaling was used to minimize the chance that service will become resource starved under high load. We can apply pod autoscaling and cluster autoscaling here. If we add more sites into production, we just have to create new deployments and they won't affect the existing sites. In addition, the key features of Kubernetes are: automated scheduling, self-healing capabilities, automated roll outs and roll backs, and horizontal scaling and load balancing.

**High Availability**. The Azure Load Balancer distributes the requests among multiple backend services. The load balancer uses health probes that determine the health of instances in the backend pool. When a health probe fails to respond, the load balancer stops sending new connections to the unhealthy instances. Existing connections are not affected, and they continue until the application terminates the flow. The ingress controller provides configurable traffic routing. By using this, a single IP address can be used to route traffic to multiple services in the K8s cluster.

**Resiliency**. Using AKS gives the system the ability to recover from failures. If there’s a pod that failed, K8s will automatically spawn a new pod or restart it. Azure Database for MySQL a managed service that delivers built-in high availability with no additional cost,  Scale as needed within seconds, Secured to protect sensitive data at-rest and in-motion, Automatic backups and point-in-time-restore for up to 35 days. With that, the developers can simply focus on the application development. The Open Service Broker for Azure (OSBA) allows the K8s cluster to connect to managed services such as Azure DB for MySQL.

**Manageability**. CI/CD pipelines for the infrastructure and the code were designed so that the IaC (ARM) templates and source codes will be in version control system. The architecture uses Azure Pipelines for the build and deployment processes. With a good CI/CD process, the team can build and deploy services independently without disrupting the other teams. Certain environments can be used for validation before deploying the new version of the services to production. A monitoring system helps prevent critical incidents that might affect the services. The Azure Monitor will be used for the monitoring system of the infrastructure. It collects, analyzes and acts on telemetry from the cloud and on-premise environments. It helps understand how applications are performing and proactively identifies issues affecting them and the resources they depend on. The Grafana instance provides visual dashboards of the application performance based on the data from Azure Monitor.

**Security**. Azure Role-based access control (RBAC) provides access management of the Azure resources through role assignments. The Azure Active Directory helps users sign in and access the Azure resources. By using Azure Key Vault, there will be a centralized control of secrets and centralized key management. Azure DB for MySQL has enterprise-grade security and compliance.