Technical Test

Problem Description

As a senior member of the cloud infrastructure team for a digital healthcare service provider you have been tasked with designing and provisioning the resources required to support a new online digital service. The service will require the creation of new infrastructure to support the web-based application that will be accessed by both patients and health care professionals. The following requirements statements have been provided by the product team.

- The implementation requires 2 separate applications:
    - A web application that provides the interface for the patients and health care professionals
    - A set of APIs that are implemented as a separate API application
- The following other resource requirements are defined:
    - The API application requires its own SQL database to store application specific data
    - Both the API application and the web application need to be able to read and write binary data such as images and PDFs into containers within a storage account
    - Both the web application and API application will need to store and retrieve sensitive application configuration values needed during application use
    - The web application needs to deliver email messages to end users
- The web application will be accessed by patients over the public internet on desktop and mobile devices
- The API application will only be accessed by the web application via server-side calls and should not be publicly accessible or discoverable

The lead security engineer and compliance manager have also provided the following requirements and general practices that must be adhered to for any presented solution.

- All public facing web endpoints must be protected from malicious actors, ensuring that the web application is protected from common threats
- Telemetry and log information must be retained for a period of at least 90 days that allows investigation of any incidents
- No service that holds data or configuration will allow unauthenticated access
- No service that holds data or configuration will have an unsecured public endpoint and must be either network restricted or deployed such that it has no defined public endpoint and can only be accessed by our deployed services
- Binary data held for the application within storage accounts can only be held for a maximum of 180 days, after which we are legally required to remove the data
- Where a technology or resource (such as a database, storage account etc) supports a “secret-less” authentication mechanism this must be used unless a clear rationale is agreed for not doing so
- We always deploy solutions using end-to-end HTTPS encryption such that no services are accessed using un-encrypted communication channels, even within private networks.

Tasks

We would like you to consider the following tasks:

1. Provide an infrastructure solution diagram(s) that shows how you would propose to provision Azure cloud resources to support the requirements provided above, your diagram should attempt to include as many of the key resources you would propose using and should account for:
    a. Resources related to management and logging
    b. Resources related to connectivity such as networking
    c. Resources related to meeting any security/compliance requirements
2. Provide any commentary and assumptions that support your solution separately.
3. Include an example of how you would provision a portion of your proposed solution using one or more code templates written using either ARM or preferably Bicep, using this as an opportunity to showcase how you would address similar requirements working within our team.
4. [Bonus] Ensure your infrastructure templates support deployment to multiple environments and regions such as “dev” and “test” across both “UK South” and “UK West”.

The information you provide will form a key part of a technical interview where we will review and discuss your approach to assess your experience, knowledge and understanding of the scenario presented.

Solution Description

Brief

As with all architectures in the modern cloud, there are usually multiple solutions available for any specific need, and combining these can lead to hundreds of different architectures that all achieve a goal. In order to fit the requirements of the challenge, I have selected an architecture focused on:

- Simplicity
- Maintainability
- Separation for application tiers (front & back end)
- Security

The proposed solution is based on Platform as a Service resources, with a SDLC managed from Azure DevOps. In this document I describe the assumptions, provide a diagram and include information on how each point from the Problem Statement is addressed.

A small sample of Bicep code is linked with a Powershell script used to convert this to ARM templates for deployment. This is now unnecessary but originally I developed it when Bicep was not in General Availability status in Azure, and I wanted to include some form of Powershell in the code base. There is also the YAML CI/CD pipeline file included ready for upload to ADO.
The templates easily support changing the region and environment; a new environment can be defined as an additional stage using the same artefact in the cicd.yml file, with conditions on the previous deployment's step. The region is parameterised for the Bicep build.

Assumptions

I have selected Azure as the only valid cloud provider due to the request of showing a portion of code in ARM or Bicep, and the role being for an Azure Cloud Infrastructure Engineer.
In this solution, I have assumed a standard landing zone deployment has taken place. A hub and spoke networking model is followed, with already existing VNet Peering between the hub and the spoke. This solution will use a peering between two VNets but these are not of separate spokes - the backend VNet should be considered a 'spoke of a spoke'.
Typically for a smaller application like this, we would expect to reuse an existing firewall, leverage identity services that were configured prior to this application, and agreements with Microsoft would already be in place for a subscription with appropriate quotas scoped ahead of the project.
I also have assumed an existing Azure DevOps exists and so basic setup of this is not required, only a new repository, pipeline and permissions with service principals are required to the destination subscription(s).
In addition to the setup of Azure DevOps, I assume self hosted agents exist in the Hub virtual network and will have full access to the deployed resources to properly deploy & configure the environment(s).
I expect, due to the requirements, that other team members that will also maintain the application have a similar skillset and are capable of DevOps deployments, interpreting logs and troubleshooting on this technology stack.

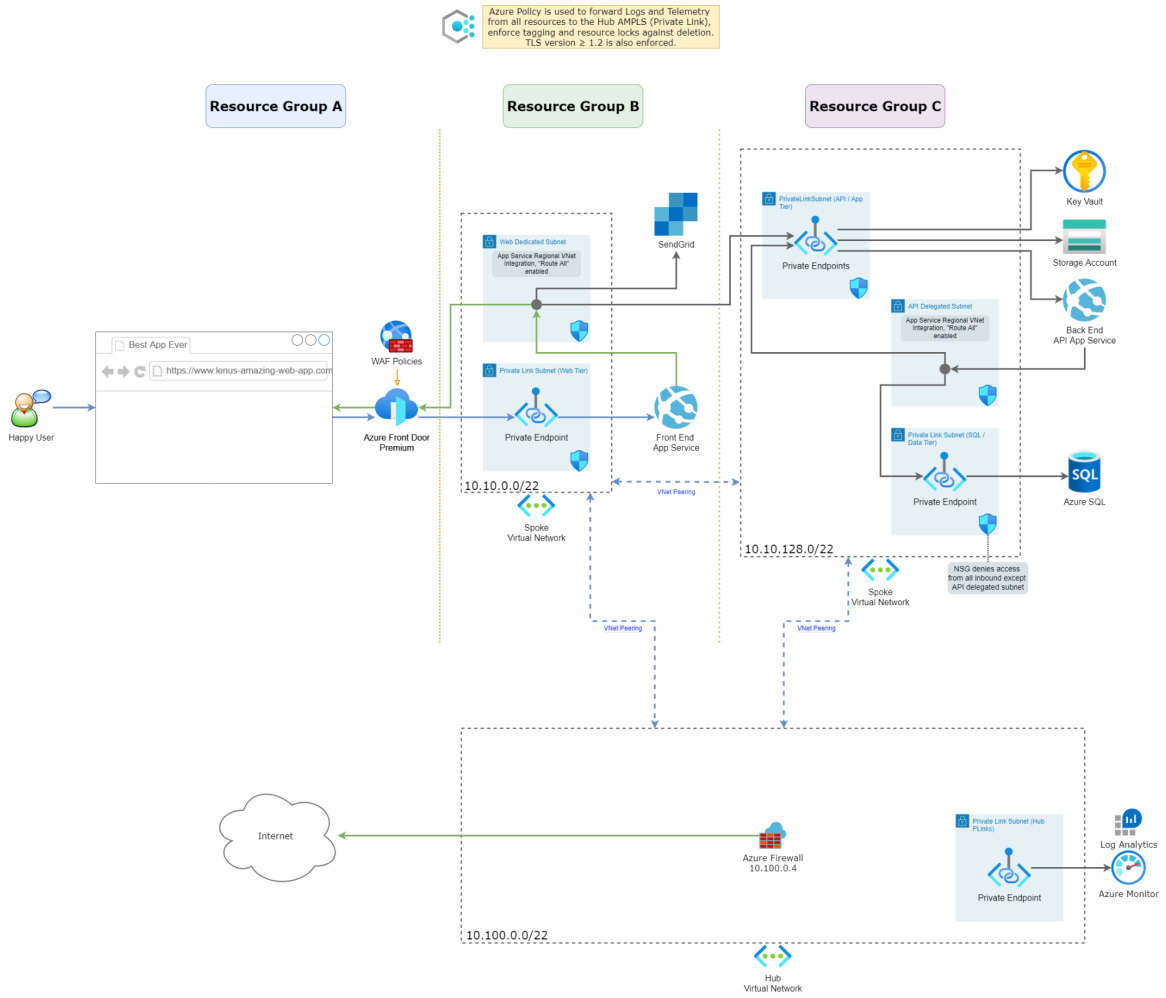
Without these presumptions, there is room for an expanded brief and more initial setup configuration steps.

Other options could include:

- Using App Service Environments to deploy the app services and a managed SQL instance as the database engine. Both of these services are natively deployed within a VNet, so there is no need for VNet Integration or Private Endpoints. As these are single-tenant isolated deployments, they cost more.
- Using the same ILB ASE to deploy the front-end web app and API app, and instead to make the front-end accessible via an application gateway
- Using Azure service endpoints in place of Private Endpoints
- Using public networking with access restrictions, relying instead on identity security
- IaaS (VM based solutions)
- Container-based solutions
- Including queuing services between the front and back end services
- Serving content via cache services and/or CDNs
- Application Gateway with WAF instead of Azure Front Door

Each of these have advantages and drawbacks, but do not fit the problem description as cleanly as this proposed solution, and in many cases will take a longer time to plan and deploy.

Diagram



Network connectivity for access to the private endpoint in the Hub is not shown as all resources need access to this and it does not add value to the diagram in clarifying how this routing works.

Details of implementation

Azure Front Door Premium is used to connect to Private Link services deployed on the front end's Spoke VNet. Front Door reduces the requirements for multi-regional deployments or disaster recovery caused by regional failures in the solution.
Azure Front Door and Azure Firewall are deployed in parallel. This is for simplicity, and on the assumption an Azure Firewall already exists.
As the app has no defined SLA for RPO and RTO, we will assume GRS is appropriate for the data on the Storage Account and for SQL for regional disaster recovery.

Here is how each point is met:

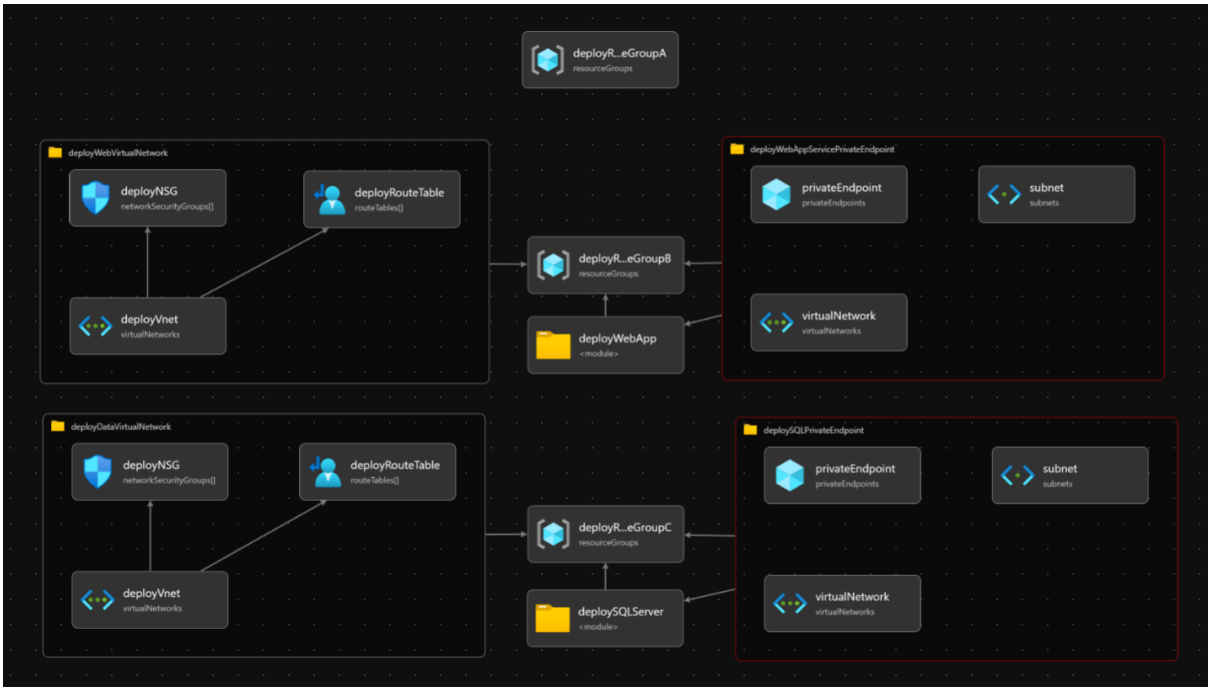
- The API application requires its own SQL database to store application specific data
    - An Azure hosted & managed SQL instance is used for the database, with the network secured by a Private Endpoint with an NSG to lock down access to only the API app service.
- Both the API application and the web application need to be able to read and write binary data such as images and PDFs into containers within a storage account
    - The Private Endpoint for the Storage Account is available to both the front and the back end App Services via a Private Endpoint, including the VNet peering for the Web Tier's App Service connectivity.
      The policy on the Storage Account grants read & write access to the Managed Identities of the App Services.
- Both the web application and API application will need to store and retrieve sensitive application configuration values needed during application use
    - A Key Vault is provided via another Private Endpoint, with an Access Policy granting the Managed Identities of the Web App Service and API App Service access to their secrets stored there, but not general access to all secrets, using the RBAC permission model.
- The web application needs to deliver email messages to end users
    - SendGrid is a managed email service in Azure, and is called via an API to send emails. In past configurations we've used a Logic App triggered by an Event Grid to send emails via Exchange Online, but since volume isn't defined I've omitted those additional components. The sender Domain needs an update to the MX record if SendGrid is not used elsewhere, usually the choice between these options is dictated by the volume of emails as Exchange Online has hard limits on frequency and recipients count.
- The web application will be accessed by patients over the public internet on desktop and mobile devices
    - Front Door provides access to the Web App Service frontend, secured by WAF policy.
- The API application will only be accessed by the web application via server-side calls and should not be publicly accessible or discoverable
    - The API application is protected by a private endpoint in a VNet that has no direct inbound routes from the Front Door. Only the Web App Service has access from outside of the VNet.
- All public facing web endpoints must be protected from malicious actors, ensuring that the web application is protected from common threats

    - A Web Application Firewall is integrated in Front Door, protecting against Application Layer threats. As this is a managed service, it benefits from Rate Limiting / Throttling, OWASP Top 10, Global WAF Policy updates and Custom Access Control.
- Telemetry and log information must be retained for a period of at least 90 days that allows investigation of any incidents
    - All services integrate with a central location by applying a policy to forward audit and application Logs to a Log Analytics workspace in the Hub, along with telemetry data sent to Azure Monitor Metrics. Access to all of this is mandated via Azure policy and is provided via the peering to the Hub VNet.
- No service that holds data or configuration will allow unauthenticated access
    - All services leverage AAD for authentication and authorisation, with the inclusion of SQL using AAD groups for DB level permissions.
- No service that holds data or configuration will have an unsecured public endpoint and must be either network restricted or deployed such that it has no defined public endpoint and can only be accessed by our deployed services
    - This is demonstrated clearly in the diagram.
- Binary data held for the application within storage accounts can only be held for a maximum of 180 days, after which we are legally required to remove the data
    - A storage account lifecycle policy is used to clear data that is untouched for more than 180 days. Given that the requirement is legal, soft delete is disabled.
- Where a technology or resource (such as a database, storage account etc) supports a “secret-less” authentication mechanism this must be used unless a clear rationale is agreed for not doing so.
    - No code contains secrets. All secrets used in the deployment are stored in the Key Vault and entered as references into the Pipeline Variables. App configuration also contains a reference to this Key Vault value - these values are generated at deployment time using a script and are then deployed to the App Services as a separate App Service Configuration resource in another deployment step. This is not present in the code as it was not part of the example Bicep.
- We always deploy solutions using end-to-end HTTPS encryption such that no services are accessed using un-encrypted communication channels, even within private networks.
    - Mandated & enforced by Azure policy, along with the option of configuring Network Security Groups to prevent non-HTTPS traffic. The SSL configuration on the App Services is set to HTTPS only.

Sample Code

https://github.com/nickcherrill/demo-webapp

The configured items are here:



Only the actual code for CI/CD pipeline and virtual networking is complete. There are more components that are not coded, but the placeholders for these files exist in the git repository:

- Front Door
- WAF Policy
- Key Vault & its Private Endpoint
- Storage Account & its Private Endpoint
- SQL & its Private Endpoint
- App Services & their Private Endpoints

It would take a few days to complete the boilerplate code for this to have a test deployment ready as a proof of concept, excluding application code. I have not included Hub resources, including:

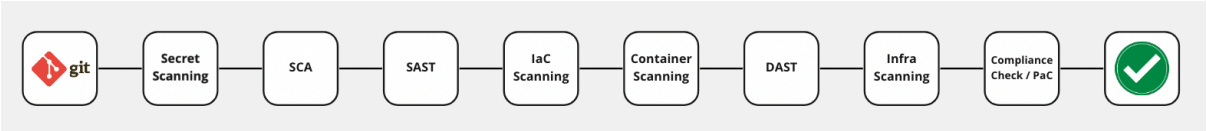
- Azure Firewall & Policy
- Virtual Network, NSG, Route Table
- Private Endpoints for Azure Monitor (AMPLS)
- Monitoring components deployed to the hub

And the Azure policy has no examples in the repository. I am able to code this and am working on something similar at the moment to pull configuration of existing policies, then deploy with any new ones back to Azure from our repositories.

Additional Considerations

Improvements to be considered include:

- Enabling CORS for developers to more easily access and test the app service hosted application
- Adding pipeline tools for ensuring IaC is following good practices, a breakdown can be found on the OWASP DevSecOpsGuideline repository



- Azure API for FHIR could be considered instead of App Services as this is a health industry application - I have no experience in using this, though
- Monitoring dashboards, alerts
- Cost control dashboards, alerts
- Caching services like Azure CDN and looking into higher performance database options like Redis, as modern web apps are expected to have extremely fast load times
- Regional storage synchronisation for the Storage Accounts and SQL servers in a Hot - Cold deployment to speed up recovery from failures, rather than using Azure's restoration process from Azure Backup
- Branch protection policies in Azure DevOps to ensure merged PRs are properly reviewed
- More time spent refactoring and improving with help from an experienced team!

Component Resources

- Azure Front Door

Used to manage at-scale edge traffic, and to increase performance for end users by presenting endpoints all around the world. This technology is cloud-native, which doesn't require any licensing; you only pay for what you use. In this workload scenario, Azure Front Door serves as the ingress point for all traffic to the consumer health portal. Rather than using Application Gateway, we use Front Door here as a decentralised, out-of-VNet option, which simplifies VNet design.

- Azure App Service

Used to host HTTP-based web services. It supports a wide array of languages, can run on Linux or Windows, fully integrates with CI/CD pipelines, and can even run container workloads as a PaaS offering. App Service allows for both scale-up and scale-out, in addition to having native integration with identity, security, and logging services in Azure. It is able to meet the scaling needs of the consumer health portal while maintaining compliance. In this architecture, it hosts the front-end web portal and back-end API as separate App Services integrated in their own virtual networks.

- Azure Log Analytics

An Azure Monitor Logs tool, which can be used for diagnostic or logging information, and for querying this data to sort, filter, or visualise them. This service is priced by consumption, and is perfect for hosting diagnostic and usage logs from all of the services in this solution.

- Azure Application Insights

Another feature of Azure Monitor, is the native Application Performance Management (APM) service in Azure. It can be easily integrated into the front-end and back-end App Services to enable live monitoring of the applications. Application Insights can easily detect performance, usability anomalies, and faults directly generated from the applications themselves, and not just from the compute platform hosting them.

- Azure Monitor Metrics

Another feature of Azure Monitor, Metrics, collects numeric data from monitored resources into a time-series database. Metrics are numerical values that are collected at regular intervals and describe some aspect of a system at a particular time.
Metrics in Azure Monitor are lightweight and capable of supporting near-real-time scenarios. For these reasons, they're useful for alerting and fast detection of issues.

Other components, briefly:

- Azure Subscription - billing logical separator
- Azure Resource Group - logical separator used for permission boundaries / SOD / application or environment separation
- Azure Private Endpoint - provides a private link from a virtual network to a PaaS resource
- Azure Storage Account - used for binary data storage
- Azure Key Vault - used for credentials and secret application configuration
- Azure Managed SQL - used for SQL database backend / queryable table based store

- Azure Virtual Network - functions similar to a traditional VLAN, this provides a virtual local network in Azure to secure, monitor, route and ensure traffic between other components
- Azure Network Security Group - restricts network access based on rules
- SendGrid - a Microsoft partner that provides a scaling email sender with advanced analytics and reporting
- Azure Policy - assists in auditing and/or enforcing settings across the scope an initiative is assigned to (from Management Group to individual Resource, as applied)