Question 1:

To improve the mentioned design, I have thought of few changes that can be done to the existing design.

- What could be done fastest?
    - Implementing proper security groups: We must set proper network security groups to handle the inbound and Outbound traffic.
    - A VPN gate way can be set to ensure a secure access to/into the VM.
    - Setting up proper rules, policies and assigning appropriate access to the resources for the team members.
    - Optimization of CI/CD:
        1. Usage of branches for different environments (development, staging and production), ensuring that the source code and config files are stored in any Version control systems like GIT, etc…
        2. Usage of CI tools like Azure Devops or Jenkins to automate the build and test the code which is pushed into a Version control system.
        3. Ensuring that the CI/CD pipeline is running the automated tests perfectly after each code commit.
        4. Using Infrastructure as code tools like Terraform, Azure resource Manager templates or Ansible, etcc… to automate the provisioning of the environment.
    - Containerization of the Microservices. Instead of building separate microservices, they can be containerized using Docker and deploy these containers using orchestration tools like Kubernetes. This helps to increase the scalability.
    - Integration of the 3rd party service cloud providers by secure architecture.
    - Separate/Migrate the DB Server to an Azure SQL database/RDS/DynamoDB for Scalability and security.
    - Implementation of Disaster recovery plan with the VM.
    - Setting up Alerts for performance metrics, security, and any other operational issues and notify the team. Real time monitoring and perform Log analytics using tools like Azure Monitor, Prometheus or Grafana.
    - Documenting the CI/CD pipeline, and other processes(deployment) and procedures(testing) and training the team members about how to maintain the pipelines.
    - Testing: Ensure proper testing methods, both Manual and Automated testing methods are used. Follow a proper workflow for the approval.
    - Perform a review on the pipeline and make some changes to the pipeline based on the feedback.

What would be your dream design?

- Containerization of microservices using tools like Docker
- Setting up proper policies and rules for the team members.
- Deploy the containers using Azure Kubernetes/Amazon EKS.

- Build CI/CD pipelines using Azure Devops/AWS DevOps.
- Building a serverless architecture for better resource management
- Setting alerts for logging purposes and real-time monitoring using Azure Monitor/Prometheus/Grafana.
- Deployment of microservices on multiple availability zones/regions.
- Setting up a disaster recovery plan as a backup.
- Setting autoscaling and load balancers for reliability.
- Migrate the data to Azure SQL database/ RDS/DynamoDB for scalability.
- Using Infrastructure as a code tools like Terraform to deploy various resources for infrastructure provisioning and management.

Pros:
- Enhanced security
- Improvement in automation and consitency
- Increased scalability and portability
- Leaverage of Built-in integrations with Azure DevOps/AWS DevOps
- High Avalability of Microservices across multiple regions.
- Usage of cloud services for better strategies and management.
- Enhanced Database performance.

Cons:
- Relies on Windows VM for Build agent.
- Existing infrastructure
- May not address long term scalability.
- Migration and Azure/AWS integration (Database, CI/CD pipelines, etcc…)
- Monthly cost for cloud services usage for maintaining the infrastructure, Azure database services/AWS services, etc…
- Compexities associated with monitoring and optimization of the logs or insights.
- Recover the pattrens within the microservices.

Which Technologies would you use ?
- AKS (Azure Kubernetes services)/Amazon EKS for container Orchestration
- Azure SQL database/RDS/DynamoDB
- Azure DevOps/Amazon DevOps for CI/CD
  Terraform to deploy various resources in Azure/AWS.
- Docker for Containerization
- Azure Monitor or Azure Log Analytics or CloudWatch for Monitoring application

What type of monitoring?
- For container and application-level monitoring, I prefer to use Prometheus and Grafana
- For Infrastructure and application-level monitoring, I will use Azure Monitor/CloudWatch.

- To track the insights for microservices performance and tracking I will use Application Insights which is an extension of Azure Monitor/CloudWatch.

Question 2:

As described in the flowchart of the infrastructure, the two environments have different strengths, pros, and cons. There can be few improvements done to that environments to enhance their benefits.

Describe what are the good points?

Coming to the architecture of the first environment:

- It has two Source control systems TFSVC and GIT which are hosted on a Azure DevOps server Classic.
- Two build agents, one for Windows and one for MaC are present. Windows build agent is connected to Fileshare A and Appstores whereas MaC build agent is directly connected to appstores.
- This environment supports for both Windows and MaC OS based developments.

Good Points about the environment is that:

- Flexibility with Version control systems TFSVC and GIT. It is up to the developers to use their preferred version control systems.
- Although the cloud platform used here is an old one (Azure DevOps Server Classic), but it can get things done for deployments, builds, manage source code, etc…
- A different build agent for Mac OS and Windows. This helps to deliver applications to the users based on the OS they use.
- As MaC build agent is directly connected to the AppStores, a direct connection is established which eliminates any additional steps to upload the applications to the Appstores.
- Developers can leverage files and dependencies stored in fileshare A during the build process, simplifying access to shared resources.
- Having separate build agents for MaC and Windows helps to build and deploy applications parallelly, faster delivery of software updates and increased efficiency.

Coming to the architecture of the second environment:

- SVN- an older version control system is integrated with Bamboo.
- Only one build agent that supports Windows applications and is connected to Fileshare B.

Good Points about the environment is that:

- Simple build environment. (Only Windows agent)
- Fileshare B provides easy access and convenient way for Storing resources and build artifacts.
- Small infrastructure and easy to troubleshoot any issues.
- Modern CI/CD and workflows with Bamboo. SVN integrated with Bamboo provides build automation capabilities.
- Low operational costs as there is only one build agent and one fileshare.
- Based on SVN commit messages, Bamboo can assist in generating changelogs or release notes.
- Supports multiple build-tools like MAVEN, ANT or even custom scripts for own project.
- Bamboo supports multiple workflows for the projects. (Trunk-based and Branch-based deployments)

Describe what you don't like about it?

First environment:

- Older version of Cloud. (Azure DevOps Server Classic)
- Dependency of Windows build agent on Fileshare A which may impact reliability and performance.
- Separate build agents require special attention towards its maintainece.
- Improper configuration may raise security concerns between the build agents and Appstores.

Second environment:

- Older version control system is being used.
- Dependency of Windows build agent on Fileshare B which may impact reliability and performance.
- SVN when integrated with modern CI/CD systems may not have that much compatability when compared to other version systems like GIT.
- Scalability challenges may arise as the entire system is relied on a Single build agent.
- Containerization and Orchestration might be challenging.
- SVN and Bamboo have a limited number of extensions and plugins results in limited support and resources.
- Might me challenging when handling complex and advanced workflows.

Describe how would you like to improve it? Suggest benefits gained by your suggested improvements?

For the first environment:

- Migrate to the latest version of Cloud services. This helps in the better scalability, updates, features, etc…
- Implement secure authentication tokens such as API keys to get access to the Appstores.
- Provide accurate permissions and set the access to the appropriate team members to avoid any security issues.

- Containerization with Docker provides very lightweight and isolated environment for the applications. It helps to manage the dependencies within the container image.
- Containerization with Docker enables easier integration with Orchestration tools like Kubernetes.
- Integrate automated testing and analyze code quality for bugs or any other security threats. Integrating with build pipelines helps to identify and resolve issues.
- Instead of two build agents, consolidate into one that supports both Windows and MaC OS which helps to reduce the number of resources required and also easy to maintain. It also helps to get quicker updates for both Windows and MaC.

For Second Environment:

- Migrate from SVN to Git for enhanced support features and plugins, better integration with modern CI/CD systems.
- Develop a migration strategy to migrate from SVN repo to git.
- Increase the build agents to improve build queue times which helps for reliable and faster builds.
- To ensure high availability, Introduce redundant build agents and artifact repos.
- Containerization with Docker provides very lightweight and isolated environment for the applications. It helps to manage the dependencies within the container image.
- Containerization with Docker enables easier integration with Orchestration tools like Kubernetes which helps to scale the containerized build agents correctly.
- Implement parallel workflow in CI/CD pipelines just as the first environment.
- Train team members about the new tools and develop documentation regarding the new environment setup and maintenance.

What are the drawbacks?

First environment:

- Migrating from Azure DevOps server classic to another cloud provider services like Azure DeVops services/AWS DevOps services involves lot of work training the team members which involves in learning new concepts and workflows.
- Make adjustments and integrate the existing tools with the new services may cause disruptions in the existing workflow.
- Increase Operational costs.
- Setting up the Azure/AWS services, configuring YAML pipelines may require lot of time and resources.
- Migrating all the data, repos, build and release definitions is challenging.
- Implementing continuous monitoring and efficient practices to ensure that the services are working efficiently or not.

Second environment:

- Migrating from SVN to Git involves lot of work like training the team members, do adjustments to the workflows, repo migration, etc…
- Rewriting the scipts, integrations that are built on SVN should be replaced to be fit the Git. Not only this, integrating with the CI/CD tools with the existing systems is also difficult.
- Risk of data loss might be possible if not handled carefully.
- Investing in new CI/CD practices results in increased operational costs.
- Ongoing projects may be delayed during this transition.
- Testing and validating the new environment to meet the company's/project's standards.

What technologies would you use and why?

- Azure DevOps Services or AWS DevOps Services can be used for CI/CD, building infrastructure and workflows. They can also facilitate modrenization, scability and efficiency for development and deployment process.
- GIT as a version control system. As it is more flexible and can be integrated with almost all CI/CD tools and other platforms.
- Distributed Artifact Repository: Implement JFrog Artifactory or similar tools for better artifact management.
- YAML Pipelines: Use code-defined pipelines for reproducibility and version control.
- Docker: Containerization for consistent and portable builds.
- Kubernetes: Orchestration for scaling and managing build agents.