Towards a Public Cloud environment for a problematic application

Technical Design

Andrei Petrov

Tech interview at Sentia, *Assessment 2*

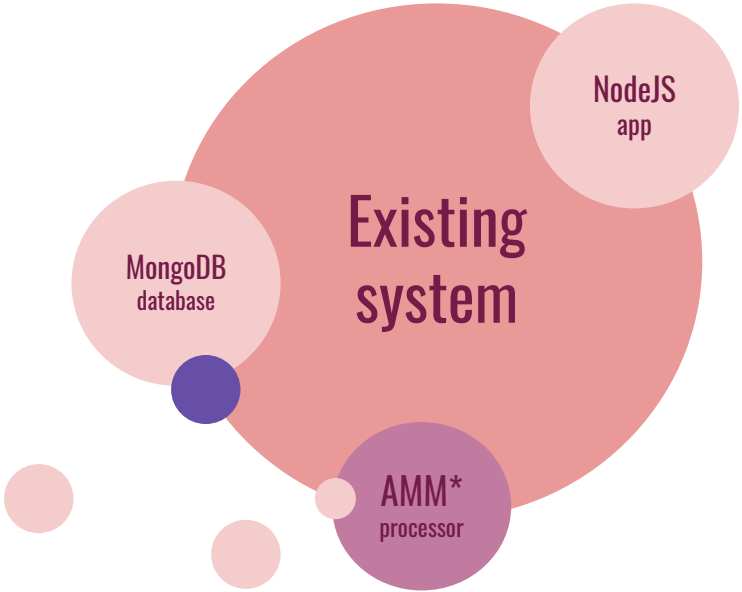
31st of August, 2018

Outline

- Motivations
- Requirements
- Design Views
- Modes of operation

Motivations

Factors influencing technical design



Situation

- One computing host contains all the application binaries
- Performance and availability issues

Needs

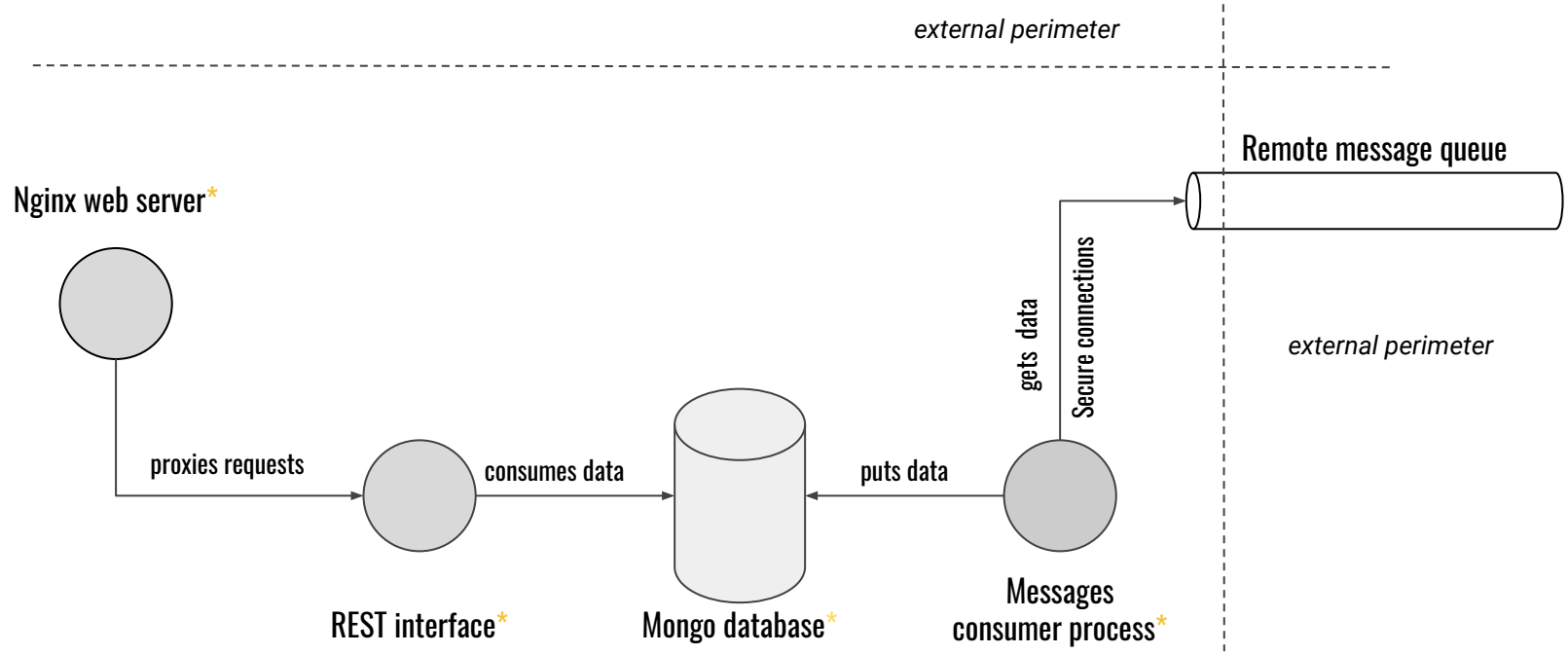
- Design a new environment to
  - Handle increases in load with no performance impacts
  - Maintain it continuously operational

Resources and constraints

- The environment has to be provisioned on Azure/AWS
- *Opt for Azure as much as possible*

ARM* = Asynchronous Messages Replicated

Motivations/2
The system architecture



* Single points of failure

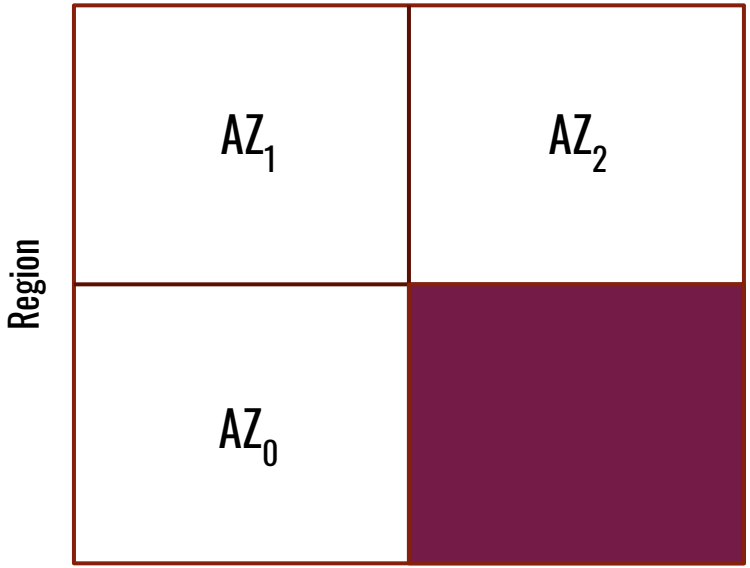
Requirements: considerations
High availability and performance

- For the customer *available* means to not lose any incoming user requests
  - Follows
    - *How can we measure availability?*
    - *How can we get initial baseline of availability for all the next improvements?*
    - *How can we monitor the availability?*
- For the customer *performant* means to be able to consume incoming request during increases in load
  - Follows
    - *How can we define application performance levels?*
    - *How can we measure and monitor the performance? Blackbox - whitebox?*
- How the availability and performance of a component in isolation impact the service in its entireness?
  - For example, if the database is overloaded

Requirements: considerations/2
High availability and performance

- We use the *separation of concerns principle* to group together logical components
  - Nginx web server and NodeJS application belong to the ***front end group***
  - The process belongs to the ***messages consumers group***
  - Each group has one responsibility
  - For example, the database: high need to evict the single point of failure

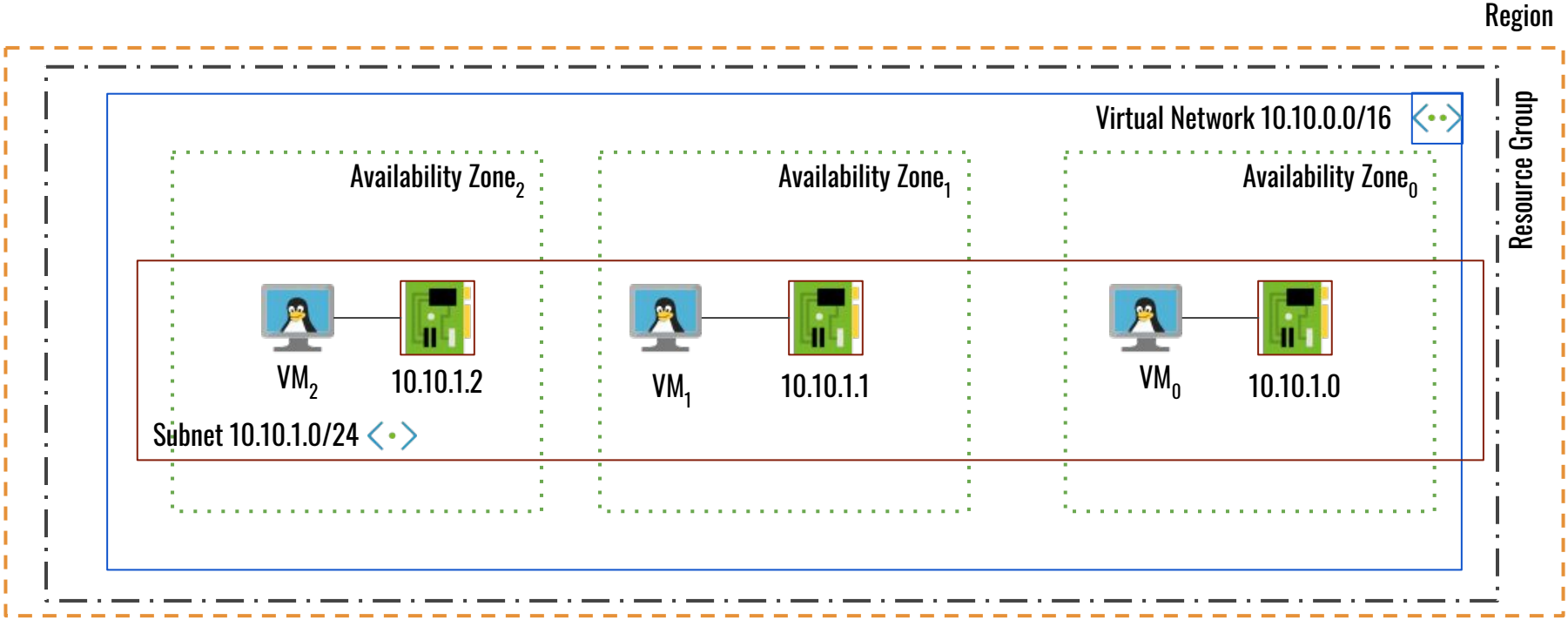
Design view
Regions and availability zones (AZs)



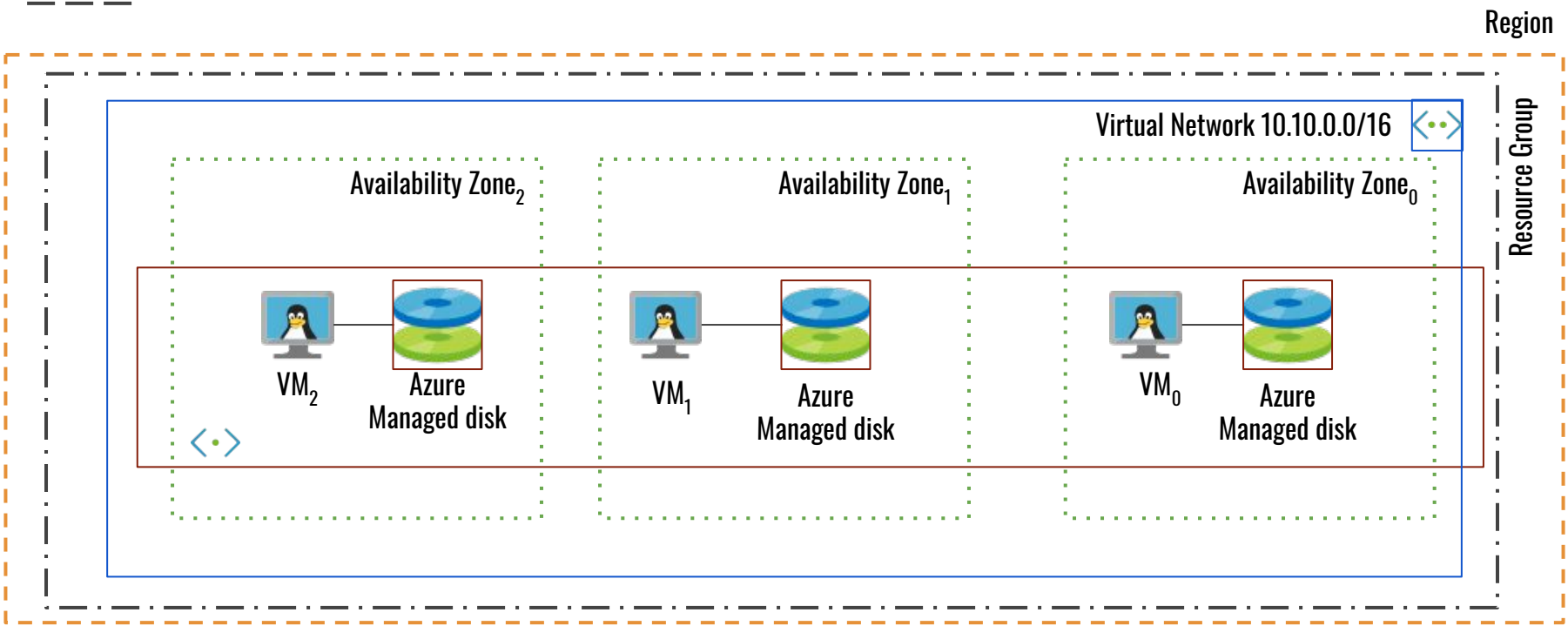
- The components distribution is paramount
  - AZs improve the service availability
  - For simple scenarios use Region resource
  - For more complex scenarios use AZs
    - For example, a cluster of 3 Etcd members
- Also distribution across Regions can be a necessity
  - Different usage scenarios
    - Region for *disaster recovery*
    - Geographical scaling
- Both options introduce security issues
- We will distributed the system components across AZs
  - The main grouping entity will be the Region

Design view

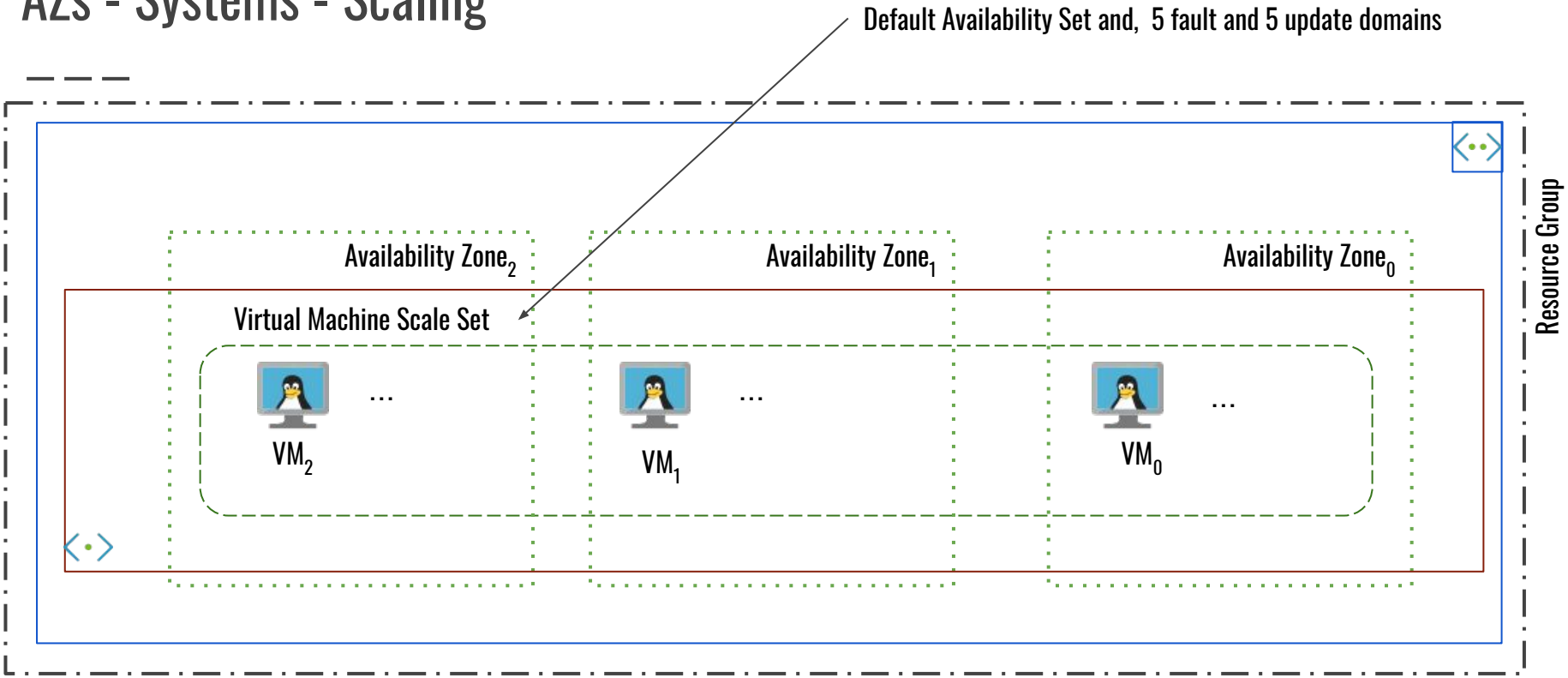
AZs - Systems - Networking



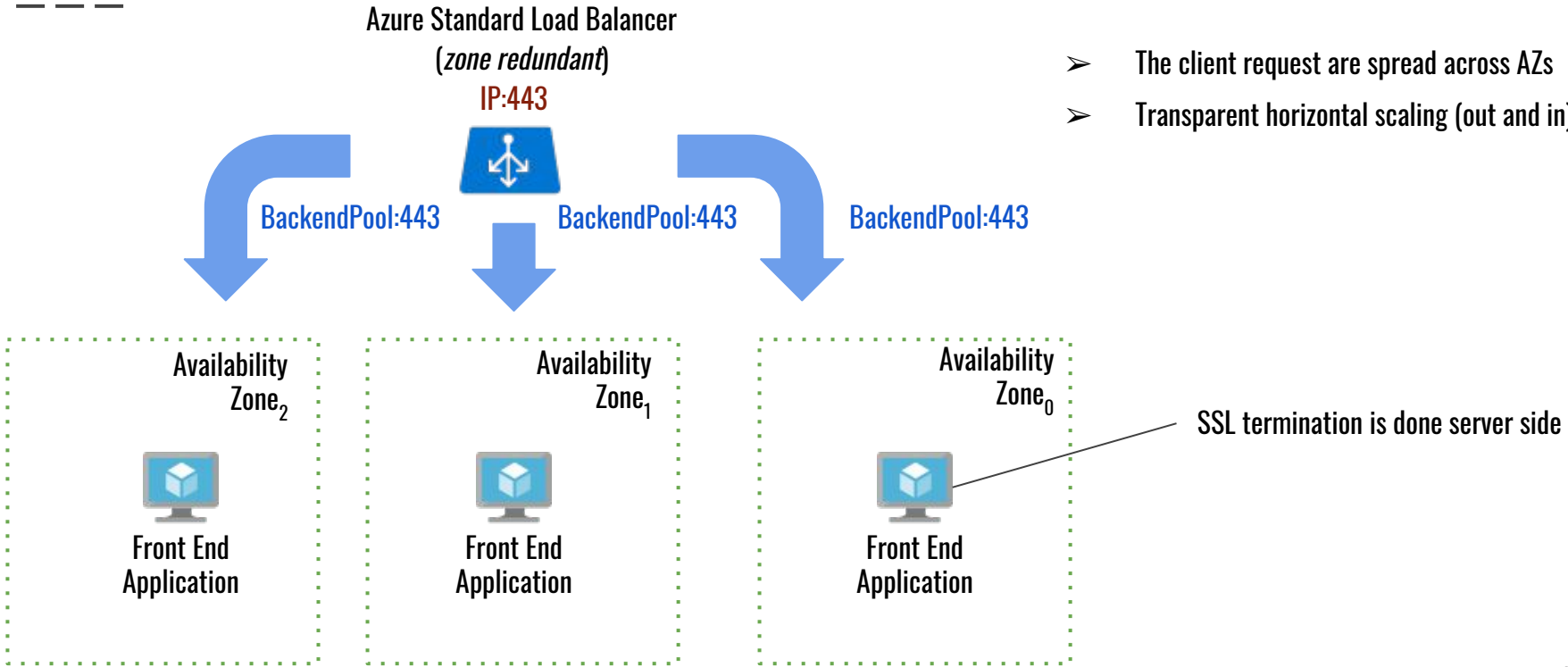
Design view
AZs - Systems - Storage



Design view
AZs - Systems - Scaling



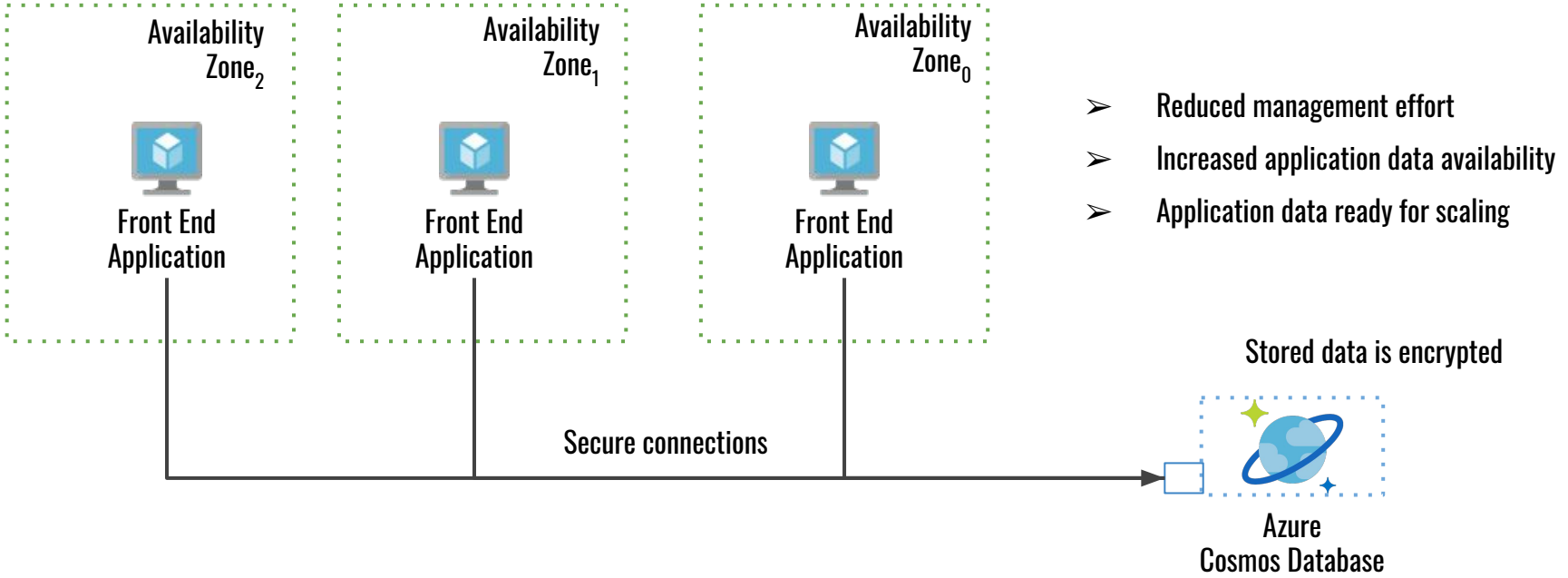
Design view
AZs - Application replicas - Load Balancing



- The client request are spread across AZs
- Transparent horizontal scaling (out and in)

Design view

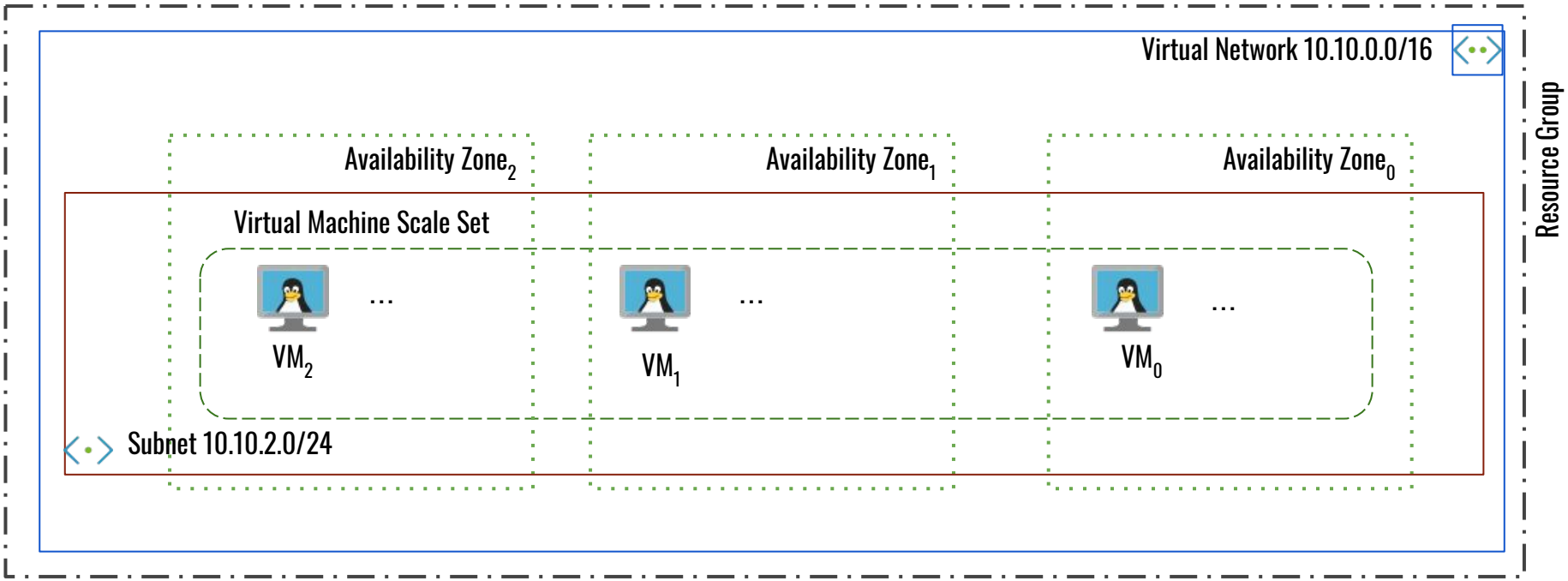
AZs - Application and database architecture



- Reduced management effort
- Increased application data availability
- Application data ready for scaling

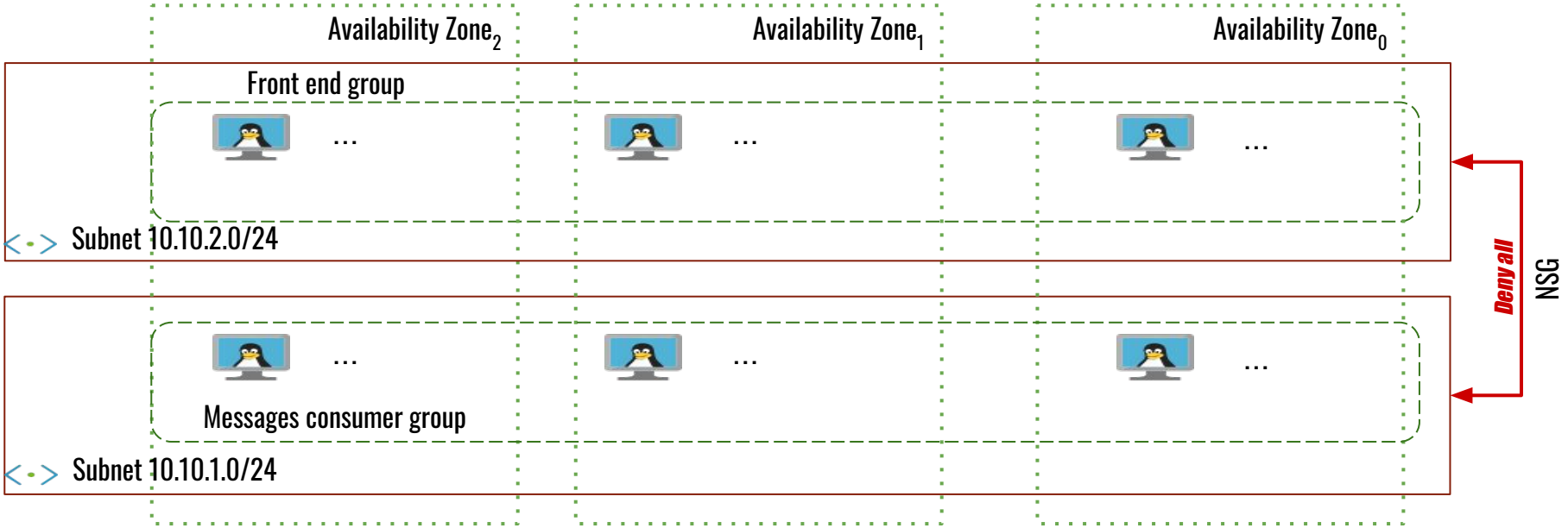
Design view

Scaling of messages consumer processes



Design view

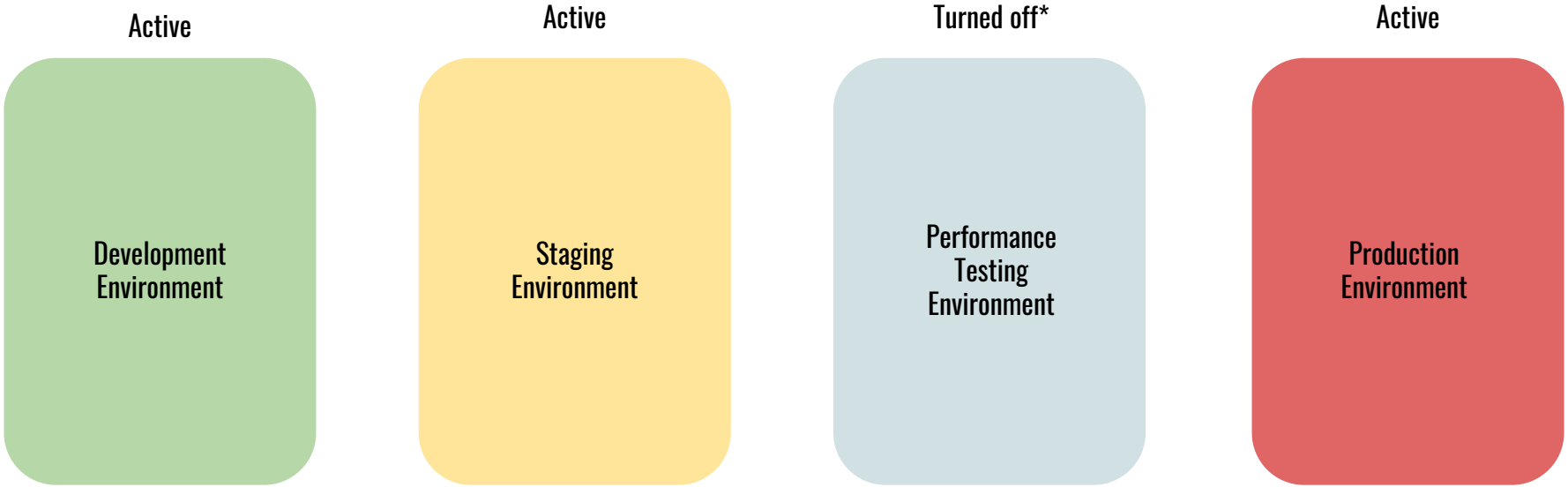
Frontend group and Messages consumer group isolation



Modes of operations
Deployment and configuration management

- Azure Resource Manager templates
  - To provision Azure resources
  - To ease the Azure resources life cycle management
- Packer virtual machine image builder and exporter
  - To create some custom virtual machine images
- Ansible or Chef (push or pull model)
  - For virtual machine configuration management
- Application deployment
  - Visual Studio Team Services
    - Offers natively support for CI/CD

Modes of operations
Required environments



Turned off*: an environment that is active only for performance testing. On-Demand.