

Software Development Essentials

Process, Structures, Tools and Practices

Development Processes

- Agile Methodologies:
 - Scrum: widely used
 - XP: developer focused (Extreme)
 - Kanban: priority based
- Continuous Delivery:
 - CI: merge code as soon as ready
 - CD: deploy frequently (daily or weekly)
 - Feature flags: control deployed but non-released features
- Mob Programming:
 - Collaborative coding
 - Knowledge transfer

People in Development

- Delivery Teams:
 - Responsible for delivering the product
 - Stream aligned teams
- Discovery Teams:
 - Explore and experiment with new ideas
 - Help design features and investigate options
- DevOps:
 - Operations tasks owned by the teams themselves
 - Infrastructure provision with self service
- More: Team Topology, Spotify Model

Technology and Tools

- Source Control Management (SCM):
 - Azure DevOps: Integrated suite of tools for managing development projects, including SCM
 - GitHub: Popular platform for hosting and collaborating on Git repositories, with features like pull requests and code reviews
 - GitLab: Comprehensive DevOps platform with built-in SCM, CI/CD, and project management
 - BitBucket: Code hosting and collaboration platform with support for Git and Mercurial repositories

Technology and Tools

- CI/CD (Build/ Test/ Deploy/ Release):
 - Azure DevOps Pipelines: Continuous integration and delivery (CI/CD) service for building, testing, and deploying applications
 - GitHub Actions: Workflow automation and CI/CD capabilities integrated with GitHub repositories
 - Jenkins: Open-source automation server for building, testing, and deploying code
 - GitLab CI/CD: Integrated CI/CD features within the GitLab platform for automating build, test, and deployment processes

Technology and Tools

- Code Quality Tools:
 - SonarQube: Continuous inspection tool for code quality and security, supporting multiple programming languages
 - Linters: Static code analysis tools for coding standards errors (ESLint, Pylint, Roslyn for C#)
- Additional Tools:
 - Docker: Containerization platform for packaging and deploying applications
 - Kubernetes: Container orchestration platform
 - Terraform: Infrastructure as Code (IaC) tool for provisioning infrastructure
 - etc

Service Integration

- Identity and Access Management (IAM)
 - Authentication/ Authorization: SSO, OAuth, OIDC, SAML, LDAP
 - Tools: Keycloak, Duende, ComponentSpace
- API Gateway
 - Handling Authentication and Reverse Proxy
 - Tools: Ocelot, Kong, Nginx, AWS, Azure
- Service Communication
 - Asynchronous: Pub-sub messages/events
 - Synchronous: Request-response through queues or RESTful APIs
 - Tools: RabbitMQ, Kafka, HTTP, Polly

Good Developer Practices

- Logs
 - Add logs to critical paths and edge cases
 - Follow standard log formats and levels
- Comments
 - What and how to comment
 - Write self-documenting code where possible
 - Use comments to explain the "why" behind complex logic or decisions
- Debugging Skills
 - Profiling, diagnostics, breakpoints, logging, benchmarking
 - Network tools, browser dev tools

Good Developer Practices

- Tests
 - Unit and integration tests for good design
 - Aim for high test coverage but prioritize meaningful tests over quantity
 - Consider Test-Driven Development (TDD) principles where applicable
- Refactoring
 - Continuous improvement of code
 - Break down large functions/methods into smaller, manageable pieces
 - Ensure code readability and maintainability
- Design and Architecture
 - Understanding the bigger picture
 - Learn and apply design patterns (e.g., Singleton, Builder, Observer)
 - Understand SOLID principles

Good Developer Practices

- Know One Level Deeper
 - Understand the abstractions you're working on top of
 - Gain knowledge of lower-level programming languages or system operations
 - Study the underlying frameworks and libraries
- Collaboration
 - Teamwork, shared knowledge, and helping others
 - Participate in code reviews and pair programming
 - Use collaborative tools effectively (e.g., version control, project management software)
- Documentation
 - Keep project documentation up-to-date
 - Document APIs, libraries, and complex algorithms
 - Maintain clear and concise doc files like README

Good Developer Practices

- Create side projects:
 - Lots of them
 - Effective way to learn
- Self-Care
 - Avoiding burnout
 - Take regular breaks and manage your workload
 - Practice mindfulness and stress-reducing techniques
- Learning and Growth
 - Continuously improve your skills
 - Attend workshops, conferences, and online courses
 - Engage with the developer community

Good Developer Practices

- Security
 - Validate and sanitize user input
 - Use encryption for sensitive data
 - Keep dependencies up-to-date and patch early
- Performance Optimization
 - Keep an eye on performance when coding
 - Profile your application to identify bottlenecks
 - Consider caching and load balancing strategies

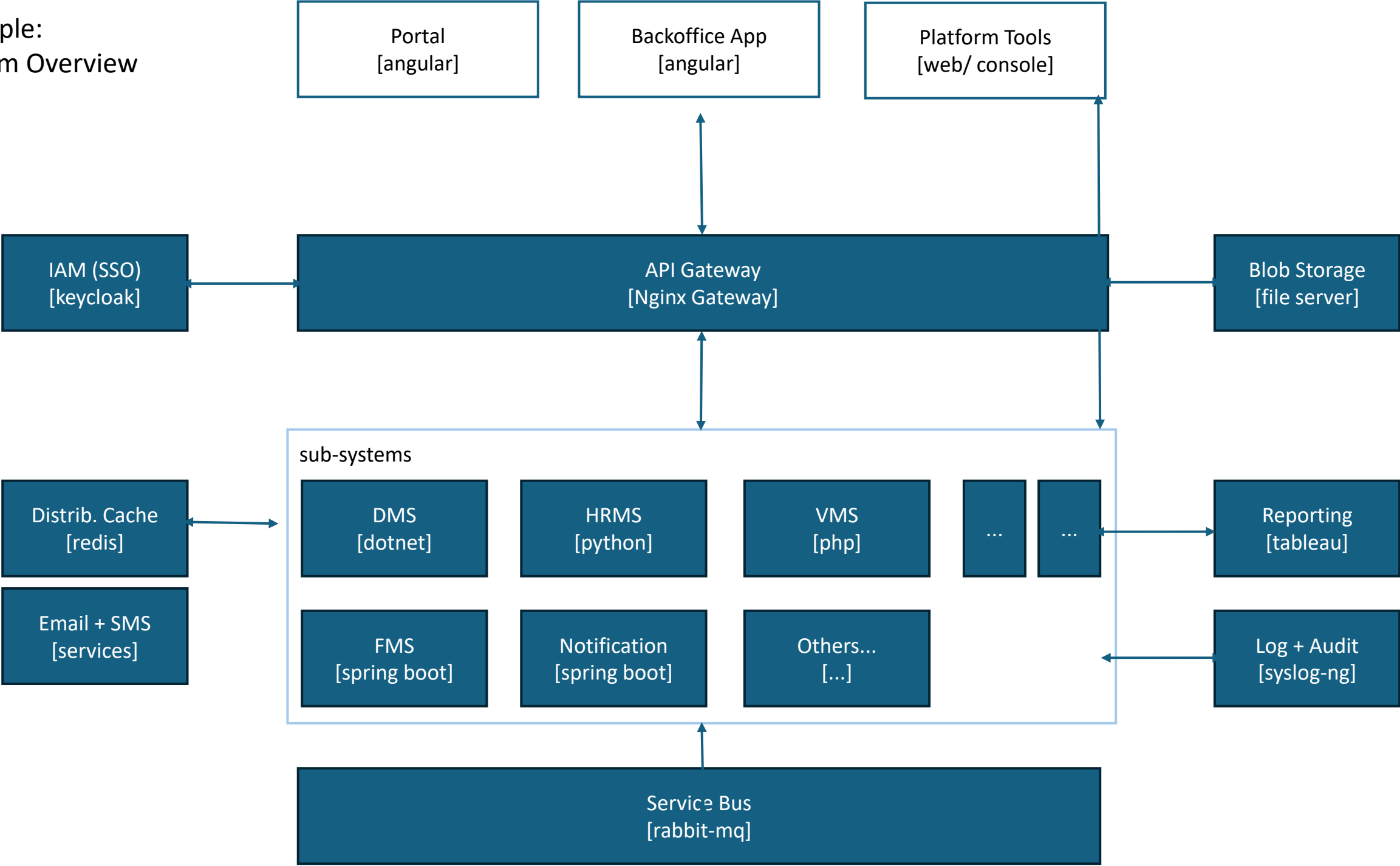
Collaboration Tools

- **Microsoft Teams:** Calendar, calls, messaging
- **Azure DevOps Boards:** SCM integration, work item tracking
- **Google Meets:** Calendar, calls
- **Cardboard:** Planning, estimation
- **Slack:** Messaging, file sharing, integrations with other tools
- **Zoom:** Video conferencing, webinars, virtual meetings
- **Jira:** Issue and project tracking, Agile development support
- **Confluence:** Collaborative documentation, knowledge sharing
- **Miro/ Lucid:** Online whiteboard for brainstorming and collaboration

Architecture (micro-services)

- Document development deployment tasks
 - C4 Model: Standard documentation for high level overview + low level details
 - Context Map: Domain model, language, boundaries
- Example:
 - **Frontend Apps**: web + mobile apps
 - **Gateway**: FE integration point
 - **IAM**: Authentication + Authorization service
 - **Micro-services**: All backend apps behind gateway
 - **Service Bus**: Internal service to service com
 - Others: caching, logging, monitoring, databases

Example:
System Overview



Vertical Slice Structure

- Features as Slices:
 - Organize code by feature rather than by technical layers
 - Each feature slice contains its folder
- Ease of Adding/Removing Features:
 - Simplifies feature management and reduces dependencies
 - Allows for better scalability and maintainability
- Separation of Concerns:
 - Each slice is self-contained and focuses on a single feature or functionality
 - Promotes single responsibility principle (SRP)
 - Enables feature-specific testing

Vertical Slice Structure

- Modular Design:
 - Encourages a modular design approach
 - Allows for easier refactoring and evolution of the codebase
 - Reduces tight coupling between different parts of the application
 - Makes it easier to replace or update components without affecting the entire system

Vertical Slice Structure

- Example [Feature: User Management]
 - **Controller**: Handles HTTP requests and responses
 - **Handler**: Contains business logic for user management
 - **Models**: Defines data structures and entities related to users
 - **Services**: Implements specific functionalities (e.g., user authentication)
 - **Repositories**: Interacts with the database