# Hotel Application – Project Blueprint

A study project to build a real-world hotel platform (think Marriott/IHG scale) using **Full-Stack .NET** with a few **Java** modules, deployable to **Azure or AWS**. This blueprint covers the scope, architecture, data model, services, APIs, DevOps, security, and a pragmatic delivery plan.

---

## 1) Vision & Goals

- **Multi-property, multi-brand** hotel platform with web and (optional) mobile clients.
- Core flows: **search → availability → dynamic pricing → book → pay → check-in/out**.
- **Operational modules**: housekeeping, maintenance, rate management, promotions, loyalty, reporting.
- **Cloud-native**: containers, IaC, CI/CD, observability.
- **Polyglot services**: majority in .NET 8; selected Java services for pricing/recommendations.

**Success criteria** - End-to-end booking runs in prod-like cloud (AKS/EKS or App Service/ECS) with real payment sandbox. - Blue/green or rolling deployments; SLOs and alerts defined. - Test coverage for critical paths (search, reserve, pay, cancel, check-in/out).

---

## 2) Personas & Primary User Stories

- **Guest**: browse, filter, view rooms, see rates, book, pay, manage reservation, check-in/out.
- **Front Desk Agent**: search reservations, check-in/out, room assignment, upsell.
- **Housekeeping**: see room status (clean/dirty/OOO), tasks, mark complete.
- **Revenue Manager**: configure rate plans, restrictions, promotions, view pickup.
- **Admin/IT**: user/role management, property setup, audit, integrations.

**Top User Stories** 1. As a Guest, I can search properties by location/dates, see real-time availability & rates, and book. 2. As a Guest, I can securely pay and receive confirmation & invoice. 3. As an Agent, I can check a guest in, assign a room, issue keys (mock), and check out. 4. As Housekeeping, I can view tasks per floor and update room status. 5. As Revenue, I can set rate plans, seasons, blackout dates, and promotions. 6. As Admin, I can add a new property, room types, and amenities with role-based access control.

---

## 3) System Architecture (High-Level)

**Frontend** - **Option A (Full-Stack .NET)**: ASP.NET Core MVC + Razor pages (public website + back-office). - **Option B**: ASP.NET Core Web API + **Angular 17** (you have strong Angular) for SPA. - (Optional) Mobile: .NET **MAUI** for front-desk tablet/guest app.

**Backend (Microservices)** - **.NET 8** (ASP.NET Core) services: AuthN/Z, Property/Inventory, Availability, Booking, Payments, Housekeeping, Loyalty, Notifications, Reporting API. - **Java (Spring Boot)** services: **Dynamic Pricing** (ML/time-series), **Search/Recommendations**.

**Data** - **Operational DB**: SQL Server / Azure SQL (or PostgreSQL on AWS RDS) for ACID. - **Cache**: Redis for availability & session cache. - **Search**: Azure AI Search / Elastic(OpenSearch) for property & room search. - **Events**: Kafka/Event Hubs or RabbitMQ for domain events (ReservationCreated, RoomStatusChanged). - **Object Storage**: Azure Blob / S3 for images, invoices, exports. - **Analytics**: Lakehouse (ADLS/S3) + Synapse/ Athena/Glue + Power BI/QuickSight.

**Infra & Platform** - **Containers**: Docker; **AKS (Azure)** or **EKS (AWS)**. Alternative PaaS: App Service/ECS Fargate. - **API Gateway**: Azure API Management / Amazon API Gateway + WAF. - **Identity**: Azure AD B2C / AWS Cognito (social login optional) + JWT. - **Secrets**: Azure Key Vault / AWS Secrets Manager. - **Observability**: OpenTelemetry + App Insights/CloudWatch, Grafana, alerts.

---

## 4) Domain-Driven Design (Bounded Contexts)

1. **Identity & Access**: users, roles, permissions, tenants (brands), SSO.
2. **Catalog/Property**: properties, room types, amenities, media, policies.
3. **Inventory**: rooms, room status, allocation, housekeeping state.
4. **Rates & Pricing**: base rates, seasons, dynamic pricing, restrictions (LOS, CTA/CTD).
5. **Availability**: calendar computation combining inventory + rates + restrictions.
6. **Booking**: reservation lifecycle (hold → confirm → modify → cancel), folio.
7. **Payments**: card tokenization, charges, refunds, invoices, PCI DSS scope reduction.
8. **Operations**: housekeeping, maintenance, tasks, work orders.
9. **Loyalty**: members, tiers, accruals, redemptions, benefits.
10. **Notifications**: email/SMS/push templates and delivery.
11. **Reports**: daily pickup, occupancy, ADR, RevPAR, housekeeping KPIs.

---

## 5) Data Model (Core Entities)

- **Property**(Id, BrandId, Name, Address, Timezone, Currency, CheckInTime, CheckOutTime)
- **RoomType**(Id, PropertyId, Name, Capacity, BedTypes, Amenities)
- **Room**(Id, PropertyId, RoomTypeId, Number, Floor, Status: Clean/Dirty/OOO)
- **RatePlan**(Id, PropertyId, Name, Description, CancellationPolicy, IncludedMeals)
- **Rate**(Id, RatePlanId, Date, Price, Currency)
- **Restriction**(Id, PropertyId, DateRange, MinLOS, MaxLOS, CTA, CTD)
- **Availability**(PropertyId, RoomTypeId, Date, Allotment, Sold, Remaining)
- **Reservation**(Id, PropertyId, Code, GuestId, CheckInDate, CheckOutDate, Adults, Children, Status)
- **ReservationLine**(ReservationId, Date, RoomTypeId, RatePlanId, Price)
- **Payment**(Id, ReservationId, Provider, Token, Amount, Currency, Status)
- **Guest**(Id, Name, Email, Phone, LoyaltyId?)
- **HousekeepingTask**(Id, RoomId, Type, Priority, AssignedTo, Status, DueDate)
- **Invoice**(Id, ReservationId, Lines[desc, qty, unitPrice], Taxes, Total)

- **LoyaltyAccount**(Id, GuestId, Tier, PointsBalance)
- **Promotion**(Id, PropertyId, Code, DiscountType, Value, Validity)
- **AuditLog**(Id, Actor, Action, Entity, Before, After, Timestamp)

  Use **row-level security** per PropertyId/BrandId for multi-tenant isolation.

---

## 6) Services & Responsibilities

### 6.1 Identity Service (.NET)

- JWT issuance, RBAC/ABAC, tenant scoping; integration with Azure AD B2C/Cognito.
- Endpoints: `/auth/register`, `/auth/login`, `/auth/refresh`, `/roles`, `/invite`.

### 6.2 Property Catalog Service (.NET)

- CRUD for properties, room types, amenities, media upload to Blob/S3.
- Integrates with Search indexer.

### 6.3 Inventory Service (.NET)

- Rooms lifecycle, status transitions, housekeeping linkage; emits `RoomStatusChanged`.

### 6.4 Rates & Restrictions Service (.NET)

- Rate plans, seasonal calendars, promotions; APIs consumed by Pricing.

### 6.5 Dynamic Pricing Service (Java, Spring Boot)

- Ingests bookings, pickup, competitor rates (mock), seasonality; outputs suggested rates.
- REST `/pricing/quote` and async `RateSuggested` events.

### 6.6 Availability Service (.NET)

- Combines inventory + rates + restrictions + pricing to compute daily availability.
- Caches per property/date in Redis. Endpoint: `/availability/search`.

### 6.7 Booking Service (.NET)

- Reservation lifecycle (idempotent); payment orchestration; overbooking rules.
- Emits events: `ReservationHeld`, `ReservationConfirmed`, `ReservationCancelled`.

### 6.8 Payments Service (.NET)

- Tokenize & charge via Stripe/Adyen/Braintree sandbox; stores tokens only (no PANs).
- Webhooks → `PaymentSucceeded/Failed` → update Booking/Invoice.

### 6.9 Housekeeping Service (.NET)

- Task board, assignments, room turns, SLAs; mobile-friendly endpoints.

### 6.10 Notifications Service (.NET)

- Templates, providers (SendGrid/Twilio), transactional emails/SMS with retries.

### 6.11 Search/Recommendations (Java, Spring Boot)

- Search over properties/rooms; “similar hotels” and upsell suggestions.

### 6.12 Reporting API (.NET)

- Pre-aggregated views (materialized) for occupancy, ADR, RevPAR, cleaning turnaround.

---

## 7) API Sketch (selected)

**Availability** - `GET /availability/search?city=Dallas&checkIn=2025-11-20&checkOut=2025-11-23&adults=2` - Response: properties with room types, nightly rates, remaining inventory.

**Booking** - `POST /bookings/hold` → blocks inventory for 15 min (Redis TTL).
- `POST /bookings/confirm` {holdId, guest, paymentToken} → reservation code.
- `DELETE /bookings/{code}` → cancel with policy validation.

**Payments** - `POST /payments/tokenize` → vaulted token (Stripe Elements style).
- `POST /payments/charge` {reservationCode, amount}.

**Housekeeping** - `GET /housekeeping/tasks?propertyId=...` | `PATCH /housekeeping/tasks/{id}` status.

**Admin** - `POST /properties` | `POST /rateplans` | `POST /restrictions` | `POST /promotions`.

---

## 8) Event Model (examples)

- `ReservationCreated {reservationId, propertyId, dates, value, channel}`
- `PaymentSucceeded {reservationId, amount, provider, txnId}`
- `RoomStatusChanged {roomId, from, to, by}`
- `RateSuggested {propertyId, roomTypeId, date, price, confidence}`

Use Kafka topics per bounded context; contract schemas in Avro/JSON Schema. Keep events immutable.

---

## 9) Cloud Deployment (Azure/AWS)

**Azure** - AKS + Container Registry; Azure SQL; Redis Cache; Event Hubs/Kafka on AKS; API Management; Storage (Blob); App Insights; Key Vault; AD B2C; SendGrid/Twilio.

**AWS** - EKS + ECR; RDS (SQL Server/Postgres); ElastiCache; MSK/Kinesis; API Gateway + WAF; S3; CloudWatch/X-Ray; Secrets Manager; Cognito; SES/SNS/Twilio.

**Networking & Security** - Private subnets for services/DBs, public only for gateway; WAF rules; mTLS inter-service (service mesh optional: Istio/Linkerd); SG/NSG least privilege; backup & PITR.

---

## 10) DevEx, CI/CD, and IaC

- **Repos**: mono-repo with folders per service or multi-repo; enforce PR checks.
- **CI**: GitHub Actions / Azure DevOps; build, test, SCA (Dependabot), SAST (SonarQube), container scan (Trivy).
- **CD**: Blue/green or canary via Argo Rollouts/Flux; approvals for prod.
- **IaC**: Terraform (cloud-agnostic) or Bicep/CloudFormation. Modules for VPC/VNet, AKS/EKS, SQL, Redis, Kafka, APIM/Gateway.
- **Secrets**: injected at runtime from Key Vault/Secrets Manager; no secrets in env files.
- **Migrations**: EF Core migrations (SQL) + Flyway for Java services.

---

## 11) Security & Compliance

- **AuthN/Z**: OAuth2/OIDC, short-lived JWTs, refresh tokens; RBAC per role; tenant claims.
- **Data**: PII minimization, encryption at rest (TDE) and transit (TLS 1.2+), RLS per PropertyId.
- **Payments**: PCI-DSS reduction via provider tokens; do not store PAN/CVV.
- **App**: input validation, output encoding, CSRF for browser flows, rate limiting, audit trails.
- **Ops**: backups, DR strategy (RPO/RTO), least privilege IAM, rotation policies.

---

## 12) Observability & SRE

- **Logs**: structured logs with correlation IDs; centralized in App Insights/CloudWatch + Loki.
- **Metrics**: RED/USE metrics; SLIs/SLOs (search latency p95 < 300ms; booking success > 99%).
- **Tracing**: OpenTelemetry spans across gateway → services → DB.
- **Dashboards**: Grafana/Kibana; on-call runbooks and alert thresholds.

---

## 13) Test Strategy

- **Unit**: xUnit/NUnit for .NET; JUnit for Java.

• **Contract**: Pact for consumer-driven contracts between frontend ↔ API and services ↔ services.
• **Integration**: Testcontainers for DB/Redis/Kafka in CI.
• **E2E**: Playwright/Cypress for web flows; synthetic checks post-deploy.
• **Performance**: k6/Locust load tests for search/booking.
• **Chaos**: fault injection (latency, pod kill) in non-prod.

---

## 14) Minimal Viable Product (MVP) Scope

1. Public website: search → availability → details → book → pay (sandbox) → confirmation email.
2. Back-office: property setup, room types, rooms, basic rate plan.
3. Housekeeping board: view tasks and update room status.
4. Java Dynamic Pricing: stubbed model that returns seasonal multipliers.

---

## 15) Iterative Delivery Plan (12–14 Weeks)

**Sprint 0 (1 wk)** – Architecture, IaC skeleton, repo setup, CI runner, code templates, auth provider.

**Sprint 1–2** – Identity, Property Catalog (CRUD, media upload), Search indexer.

**Sprint 3–4** – Inventory + Rates/Restrictions services; Redis cache; Availability API.

**Sprint 5–6** – Booking orchestration + Payments sandbox + Notifications (email).

**Sprint 7** – Frontend SPA/MVC flows for guest; booking confirmation & invoice PDF.

**Sprint 8** – Housekeeping service & board; room status transitions.

**Sprint 9** – Java Dynamic Pricing v1 (rule-based), integrate with Availability.

**Sprint 10** – Observability, dashboards, load tests, hardening (rate limits, WAF).

**Sprint 11** – Blue/green deployment, disaster recovery drill, docs.

**Sprint 12–14** – Stretch: Loyalty basics, promotions, reporting API, recommendations.

---

## 16) Technology Choices (suggested)

• **Frontend**: Angular 17 (standalone) or Razor Pages; Tailwind + shadcn/ui styling.
• **.NET**: ASP.NET Core 8, EF Core 8, FluentValidation, MediatR, AutoMapper.
• **Java**: Spring Boot 3, Spring Data, MapStruct.
• **DB**: Azure SQL or AWS RDS for SQL Server/Postgres; Redis; OpenSearch/Elastic.
• **Messaging**: Kafka/MSK or Event Hubs; Schema Registry.

- **Payments**: Stripe/Adyen sandbox.
- **Emails/SMS**: SendGrid/Twilio.
- **IaC**: Terraform; Docker; AKS/EKS; GitHub Actions; Argo CD/Flux; Key Vault/Secrets Manager.

---

## 17) Sample EF Core Schema Snippets (illustrative)

```
CREATE TABLE Reservation (
  Id UNIQUEIDENTIFIER PRIMARY KEY,
  PropertyId UNIQUEIDENTIFIER NOT NULL,
  Code NVARCHAR(12) UNIQUE NOT NULL,
  GuestId UNIQUEIDENTIFIER NOT NULL,
  CheckInDate DATE NOT NULL,
  CheckOutDate DATE NOT NULL,
  Adults INT NOT NULL,
  Children INT NOT NULL,
  Status NVARCHAR(20) NOT NULL,
  CreatedAt DATETIME2 NOT NULL DEFAULT SYSUTCDATETIME()
);

CREATE TABLE Rate (
  Id UNIQUEIDENTIFIER PRIMARY KEY,
  RatePlanId UNIQUEIDENTIFIER NOT NULL,
  [Date] DATE NOT NULL,
  Price DECIMAL(10,2) NOT NULL,
  Currency CHAR(3) NOT NULL
);
```

---

## 18) Reference Sequence (Booking)

1. Guest searches → **Gateway** → Availability Service → Inventory + Rates + Pricing + Restrictions → result cached.
2. Guest selects room → **Booking Service** creates **hold** (TTL) and price lock.
3. Guest enters details & card → **Payments** tokenize & charge → webhook returns `PaymentSucceeded`.
4. **Booking** confirms reservation, persists folio, emits `ReservationCreated` → Notifications email.
5. Housekeeping gets task to prepare room if same-day arrival.

---

## 19) Environment Strategy

- **Local**: docker-compose (SQL, Redis, Kafka), dev containers.
- **Dev**: single-AZ cluster, cost-optimized DB; feature flags enabled.

- **Staging**: 1:1 with prod; load tests; chaos.
- **Prod**: multi-AZ, backups, autoscaling, canary rollout.

---

## 20) Risks & Mitigations

- **Scope creep** → strict MVP; backlog grooming; feature flags.
- **PCI scope** → tokenize cards; hosted payment fields; no PAN storage.
- **Multi-tenancy leaks** → RLS, robust authZ checks; contract tests.
- **Cost overruns** → spot instances for non-prod; serverless where possible.

---

## 21) Backlog (MVP User Stories)

- As Admin, create a property with room types and upload images.
- As Guest, search hotels by city/dates and filter by price/amenities.
- As Guest, book a room and pay with a card; receive confirmation email.
- As Agent, view reservations and check a guest in/out.
- As Housekeeping, view and complete cleaning tasks.
- As Revenue, set base rates and restrictions.

---

## 22) Next Steps (You Can Start Today)

1. Pick cloud (Azure or AWS) and baseline stack (Angular SPA + .NET APIs suggested).
2. Scaffold repos and CI (Actions/DevOps) + IaC for network, AKS/EKS, SQL, Redis.
3. Build Identity + Property Catalog; ship dev environment; wire App Insights/CloudWatch.
4. Implement Availability → Booking → Payments → Notifications E2E.
5. Add Java Dynamic Pricing stub and integrate.

---

## 23) Appendix – Folder & Repo Structure (mono-repo example)

```
/hotel-platform
  /apps
    /web-guest (angular)
    /web-backoffice (razor or angular)
  /services
    /identity (.net)
    /catalog (.net)
    /inventory (.net)
    /rates (.net)
    /availability (.net)
    /booking (.net)
    /payments (.net)
    /housekeeping (.net)
    /notifications (.net)
    /pricing-java (spring-boot)
    /search-java (spring-boot)
  /deploy
    /terraform (environments: dev/stage/prod)
    /k8s (helm charts/manifests)
  /docs
    architecture.md
    api-specs.md (OpenAPI)
    runbooks/
  /shared
    /contracts (avro/json-schemas)
    /libs (.net shared packages, java libs)
```

---

## 24) Team of 4 — Git & GitHub Setup (Real-world)

This section gives you a plug-and-play plan for repo setup, branching, PR rules, CI, and a ready-to-run backlog with owners.

### 24.1 Roles (suggested)

- **A. Tech Lead / DevOps (You)** – repo admin, CI/CD, infra, reviews across services.
- **B. Backend (.NET)** – Identity, Booking, Availability.
- **C. Java Services** – Pricing, Search/Recommendations.
- **D. Frontend** – Angular (guest), Back-office UI.

### 24.2 Repository Strategy

- **Mono-repo** (`hotel-platform`) with folders from §23.
- **Branching: Trunk-based**
- `main` (protected): always shippable; only via PR.
- `dev` (optional fast-merge): integration branch for feature tempo.
- Feature branches: `feat/<area>-<short-desc>`
- Hotfix: `fix/<issue-id>`.
- **Tags/Releases**: semantic versioning: `v0.1.0`, `v0.2.0` …

### 24.3 Quick Start — Local & Remote

```
# 1) Create repo locally
mkdir hotel-platform && cd hotel-platform
git init -b main

# 2) Add skeleton folders
```

```
mkdir -p apps/web-guest apps/web-backoffice services/
{identity,catalog,inventory,rates,availability,booking,payments,housekeeping,notifications,pricin
java,search-java}
 deploy/{terraform,k8s} docs/shared contracts shared/libs

# 3) Create basic files
printf "bin/
obj/
node_modules/
.env
*.user
*.swp
*.DS_Store
" > .gitignore
printf "[*]
end_of_line = lf
insert_final_newline = true
charset = utf-8
indent_style = space
indent_size = 2
" > .editorconfig

# 4) First commit
git add . && git commit -m "chore(repo): bootstrap mono-repo skeleton"

# 5) Create GitHub repo then add remote
git remote add origin git@github.com:<org>/hotel-platform.git

git push -u origin main

# 6) (Optional) create dev branch for rapid integration
git checkout -b dev && git push -u origin dev
```

## 24.4 Protections & Policies (GitHub Settings)

- Protect `main` (and `dev` if used):
- Require PR, 1–2 code owners reviews, passing status checks, no direct push, linear history.
- Require signed commits (optional) and secret scanning.
- **CODEOWNERS** (root):

```
# Frontend
/apps/web-* @frontend-owner @techlead
# .NET services
/services/identity @dotnet-owner @techlead
/services/booking @dotnet-owner @techlead
/services/availability @dotnet-owner @techlead
# Java services
```

```
/services/pricing-java @java-owner @techlead
/services/search-java @java-owner @techlead
# IaC/Deploy
/deploy/** @devops-owner @techlead
```

- **PR Template** (`.github/pull_request_template.md`):

```
## What & Why

## Changes
-

## Tests
- [ ] unit  [ ] integration  [ ] manual screenshot

## Checklist
- [ ] Conventional commit title
- [ ] Updated docs/openapi
- [ ] No secrets in code
```

- **Issue Templates** (`.github/ISSUE_TEMPLATE/feature.yml` & `bug.yml`) and labels: `feat`, `bug`, `techdebt`, `infra`, `docs`, `blocked`, `good-first-issue`.

### 24.5 Commit & PR Conventions

- **Conventional Commits**: `feat:`, `fix:`, `chore:`, `docs:`, `refactor:`, `test:`, `ci:`.
- Small PRs (≤ 300 lines). Link issues: `Fixes #123`.

### 24.6 CI — GitHub Actions (minimal but real)

**.NET build/test** (`.github/workflows/dotnet.yml`)

```
name: dotnet
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-dotnet@v4
        with: { dotnet-version: '8.0.x' }
      - run: dotnet restore
      - run: dotnet build --configuration Release --no-restore
      - run: dotnet test --configuration Release --no-build --collect "XPlat
Code Coverage"
```

**Java build/test** (`.github/workflows/java.yml`)

```
name: java
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-java@v4
        with: { distribution: 'temurin', java-version: '21' }
      - run: ./mvnw -q -B -e -DskipTests=false test
```

**Container build & push** (`.github/workflows/docker.yml`)

```
name: docker
on:
  push:
    branches: [ main ]
    paths: [ 'services/**', 'apps/**' ]
jobs:
  build:
    runs-on: ubuntu-latest
    permissions: { contents: read, packages: write, id-token: write }
    steps:
      - uses: actions/checkout@v4
      - uses: docker/setup-buildx-action@v3
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}
      - name: Build & push
        run: |

IMAGE=ghcr.io/${{ github.repository }}/booking:$(git rev-parse --short HEAD)
          docker build -t $IMAGE services/booking
          docker push $IMAGE
```

## 24.7 Project Board (Kanban) & Milestones

Create **GitHub Projects** board with columns: `Backlog`, `Ready`, `In Progress`, `Review`, `Done`. Milestones: `MVP-0.1`, `MVP-0.2`, `MVP-1.0`.

### 24.8 Initial Tasks (Issues) — Assign & Estimate

Estimates use t-shirt sizes (S ≈ 0.5–1d, M ≈ 2–3d, L ≈ 4–6d).

**Infra/DevEx (A – Tech Lead)** 1. `infra` Create Terraform baseline for AKS/EKS, Azure SQL/RDS, Redis (L) 2. `ci` Setup Actions: dotnet/java/docker, required checks (M) 3. `sec` Configure OIDC to cloud, secret store integration (Key Vault/Secrets Manager) (M) 4. `docs` CONTRIBUTING.md, commit guidelines, runbooks (S)

**Backend .NET (B)** 5. `feat` Identity service: OAuth2/OIDC + JWT, EF Core migration, seed roles (M) 6. `feat` Catalog service: properties/room types CRUD + Blob/S3 upload (M) 7. `feat` Availability service: search endpoint + Redis cache (L) 8. `feat` Booking service: hold → confirm flow, idempotency keys (L) 9. `test` Contract tests between Booking ↔ Payments (S)

**Java Services (C)** 10. `feat` Pricing (Spring Boot): rule-based seasonal multiplier + `/quote` (M) 11. `feat` Search (Spring Boot): property text search + filters (M) 12. `infra` Flyway + Testcontainers for Postgres in CI (S)

**Frontend (D)** 13. `feat` Guest SPA: search/results/detail/booking steps (L) 14. `feat` Back-office: dashboard + reservations grid + room status board (L) 15. `ux` Shared UI kit (Tailwind, shadcn/ui) + Dark mode (S) 16. `test` E2E Playwright: search→book→confirm (M)

**Cross-Cutting** 17. `feat` Payments sandbox (Stripe): tokenize + charge + webhook (M) — (B/A) 18. `feat` Notifications (SendGrid): confirmation email + templates (S) — (B/D) 19. `obs` OpenTelemetry + App Insights/CloudWatch dashboards (M) — (A) 20. `perf` k6 load test for availability & booking (S) — (A)

### 24.9 Automation for Issues & Labels (optional)

Create labels quickly:

```
gh label create feat --color FFD700 --description "New feature"
gh label create bug --color D73A4A
gh label create techdebt --color 0366D6
gh label create infra --color 0E8A16
gh label create docs --color 808080
```

Auto-assign via **CODEOWNERS** + branch rules; or use `github-actions[bot]` to add reviewers based on path.

### 24.10 Environment Promotion

- **Dev** → auto-deploy on merge to `dev`.
- **Stage** → manual GitHub Environment approval + smoke/E2E checks.
- **Prod** → protected env with required reviewers; canary 10% → 100% using Argo Rollouts.

### 24.11 Secrets & Config

• Store all secrets in Key Vault/Secrets Manager; expose to pods via CSI Secret Store driver.
• App settings per env via **Helm values** or **App Service/ECS task defs**.

### 24.12 Checklists

**Before starting a feature** - Issue exists with acceptance criteria, linked to milestone & board. - Create `feat/...` branch from `dev`.

**Before merging PR** - All checks green; 1–2 reviews; migration scripts reviewed; OpenAPI/docs updated.

**Before release** - Tag version; changelog updated; smoke tests pass; rollback plan prepared.

---

## 25) Cost-Optimized Setup (Use Free/Almost-Free Wherever Possible)

Goal: keep a 4-dev study project under **$0–$20/month** outside of spikes, with production-like behavior.

### 25.1 Guiding Principles

• Prefer **managed free tiers** for public endpoints; self-host only if trivial.
• Run most workloads **locally** (Docker Compose) and **turn off cloud** outside demos.
• **One shared Dev** environment, **ephemeral preview** envs only when needed.
• Aggressive **auto-pause/scale-to-zero**; nightly **auto-shutdown** jobs.

### 25.2 Stack Choices — Free-First

**Source Control & PM** - GitHub Free (private repo, Actions minutes included); Projects (kanban), Issues. - GitHub Container Registry (GHCR) for images (avoid paid registries).

**CI/CD** - GitHub Actions on Linux runners (cheapest). Cache deps to cut minutes. Matrix builds only on `main`. - Optional: **Act** CLI for local CI runs to save minutes (dev machines).

**Frontend Hosting** - **Azure**: Static Web Apps (Free) for Angular SPA (includes free SSL, auth stubs). - **AWS**: Amplify Hosting (Free tier) or GitHub Pages (public only) + Cloudflare proxy (Free) for DNS/CDN.

**APIs / Containers** - **Azure**: **Container Apps** (Consumption; free allowance) > App Service Free (F1) for simple APIs. - **AWS**: **Lambda + API Gateway** for slim endpoints (serverless, generous free) or **ECS Fargate** only when needed. - Local dev via **docker-compose** (SQL, Redis, Kafka, services).

**Database** - For MVP, use **PostgreSQL** or **SQL Server Dev** locally. - Cloud: **Postgres serverless/burstable** with **stop/start** (Azure Flexible Server Stop, AWS RDS stop) to avoid 24/7 billing. Or use **Neon**/**Supabase** free tiers for Postgres (non-PII).

**Cache/Queue** - **Redis**: Local container for dev; cloud cache only if needed (tiny tier) or free **Upstash**. - **Events**: Use **RabbitMQ**/Kafka locally; skip managed Kafka early (costly). For cloud, use **SQS/SNS** (cheap) or Azure **Storage Queues**.

**Object Storage** - **S3/Azure Blob** – pennies; lifecycle rules to auto-expire uploads in dev.

**Auth** - Start with **ASP.NET Identity (JWT)** (no external cost). Social OAuth (Google/GitHub) is free. - Avoid paid IdP (B2C/Cognito MAU fees) until needed.

**Payments** - **Stripe** in **test mode** (free). Keep it in test for the study project.

**Email/SMS** - **Email**: Brevo (Sendinblue) free daily limit, or SES free tier (if behind EC2; else low cost). Start with **development SMTP** (MailHog) locally. - **SMS**: avoid at first; use email notifications to stay free.

**Search** - Use **Meilisearch** or **Typesense** (open-source) locally; index property/room docs. Avoid managed Elastic until needed.

**Logging/Monitoring** - **OpenTelemetry** → **Grafana Cloud Free** (metrics/logs/traces small quotas) or self-host **Prometheus + Grafana + Loki** in dev only. - Basic uptime checks: **GitHub Actions cron** + curl smoke tests.

**Maps/Geo** - **MapLibre + OpenStreetMap** tiles (free), avoid Google Maps fees; cache responsibly.

**Feature Flags** - **Unleash OSS** (container) or a simple config flag service; avoid SaaS cost.

## 25.3 Cost-Guardrails

- **Budgets & Alerts**: set $10 monthly budget with email alerts (Azure/AWS Budgets).
- **Auto-Shutdown**: nightly job to stop RDS/Flexible Server; scale Container Apps to zero.
- **TTL Policies**: S3/Blob dev buckets delete after 7 days; log retention 7–14 days.
- **Preview Envs**: only for PRs labeled `needs-preview`; destroy on merge/close.

## 25.4 Minimal Cloud Footprint (Dev)

- **Frontend**: Static Web Apps / Amplify (free)
- **API**: 1 Container App or 2–3 Lambdas for `availability`, `booking`, `payments` (serverless)
- **DB**: Postgres serverless paused (incurs near-zero when off)
- **Storage**: One dev bucket/container (7-day TTL)
- **Email**: Brevo free plan (or SES sandbox)

## 25.5 Backlog Additions (Free-Tier Work)

- `infra` Budget + alerts + auto-shutdown scripts (A – Tech Lead)
- `infra` Container Apps/Lambda IaC with scale-to-zero (A)
- `feat` Replace AD B2C/Cognito with ASP.NET Identity + OAuth social (B)
- `feat` Email via Brevo/SES + local MailHog (B)
- `feat` Meilisearch container + indexer job (C – Java or B – .NET)

- `obs` Grafana Cloud Free + OTEL collector (A)
- `docs` Cost runbook: stop/start DB, clean buckets, rotate logs (A)

### 25.6 Example IaC Toggles (Terraform variables)

```
variable "use_serverless_api" { type = bool  default = true }
variable "enable_preview_envs" { type = bool  default = false }
variable "db_auto_stop_hour" { type = number default = 22 } # local time
variable "logs_retention_days" { type = number default = 14 }
variable "enable_grafana_cloud" { type = bool  default = true }
```

### 25.7 Local-First Dev Compose (snippet)

```
version: "3.9"
services:
  api.identity:
    build: ./services/identity
    env_file: .env
    depends_on: [db, redis]
  db:
    image: postgres:16
    environment: [POSTGRES_PASSWORD=postgres, POSTGRES_DB=hotel]
    volumes: [pgdata:/var/lib/postgresql/data]
  redis:
    image: redis:7
  search:
    image: getmeili/meilisearch:latest
  mailhog:
    image: mailhog/mailhog
    ports: ["8025:8025"]
volumes:
  pgdata: {}
```

### 25.8 What to Defer to Stay Free

- Managed Kafka/MSK/Event Hubs, dedicated Redis, paid IdP, SMS, full WAF, paid APM.
- Only add when the MVP is stable and there's clear need.