PIZENITH

SOFTWARE DEVELOPMENT BEST PRACTICES

Jan 2025

Version 1.0

Production Checklist: Best Practices to Follow

Prepared For

Public

www.pizenith.com Blog Repos X Info

Prepared By

Pizenith Technologies LLP

Jan 20, 2025

General Use Case

INDEX

Production Ready Checklist

Production deployment refers to the process of moving code changes from a development environment to a live production environment, making them accessible to end users. This process requires collaboration between multiple teams, including development, quality assurance, and operations teams.

The main goal of production deployment is to ensure that the code changes are deployed with minimal downtime and that the end users are not impacted negatively. This requires careful planning and execution, as well as thorough testing in the development environment, to ensure that the code changes function as expected.

General

- ☐ **Ownership**: Service owners are identified. Contact information and methods are provided.
- ☐ **Onboarding**: Integration instructions for APIs are documented.
- ☐ **Defined service-level indicators (SLIs) / service-level objectives (SLOs) / service-level agreements (SLAs)**: The SLIs and SLOs are documented and accessible. If applicable, you've also documented the SLAs.

Disaster Recovery

- ☐ **Disaster recovery (DR)**: DR plans have been documented and tested.
- ☐ **Backups**: Backups of data occur regularly.
- ☐ **Redundancy**: Services should include at least two instances and could require deployment in multiple regions or locations.

Deployment

- [ ] **Deployment strategy**: The automated deployment strategy has been documented. For example, strategies include blue-green, canary, or others to create safer zero-downtime deployments.
- [ ] **Continuous integration**: When engineers commit their changes, the system kicks off automated builds, tests, and deployment to a lower level environment.
- [ ] **Continuous delivery**: Deploying to production involves nothing more than approval and a click of a button. Changelogs and release notes indicate what changes exist in each environment.
- [ ] **Static code analysis**: Code is automatically scanned, formatted, or linted according to coding standards.

Operations

- [ ] **On-call policy**: The service has an on-call system that pages the owning team for incidents. Ideally, this involves tools like PagerDuty or Squadcast.
- [ ] **Incident management**: The incident management and escalation processes have been documented. This includes processes for postmortem and long-term remediation.
- [ ] **Runbooks**: Runbooks have been written and are accessible, with known failure scenarios. You update runbooks whenever a new scenario is uncovered.
- [ ] **Logging**: The service utilizes centralized logs, and the logs can be accessed easily.
- [ ] **Metrics**: At a minimum, the Four Golden Signals are available for the service.
- [ ] **Tracing**: The application transactions can be traced, using the appropriate tools and sampling configuration for the service.

Testing

- [ ] **Unit tests**: Unit tests execute at every code push, automatically.
- [ ] **Integration tests**: If appropriate, automated integration tests execute and pass successfully.
- [ ] **End-to-end or acceptance tests**: Automated end-to-end or acceptance tests run as part of the continuous integration / continuous deployment (CI/CD pipeline). If manual testing is required, test results are documented.
- [ ] **Broken tests**: Failing tests break the build.

Resiliency

- ☐ **Load testing**: Load tests are automated or occur on a regular cadence. You document and publish the results.
- ☐ **Stress testing**: Stress tests are automated or occur on a regular cadence. You document and publish the results.
- ☐ **Chaos engineering**: Once the applications have proven the ability to stand up to load and stress, chaos engineering is integrated to identify weak points and opportunities to reduce failures.

Security

- ☐ **Authentication/authorization**: Each service or application requires proper authentication and authorization.
- ☐ **Secrets management**: Secrets are secured properly in a vault or secret store. Tools like truffleHog or git-secrets scan code to identify potential secrets.
- ☐ **Static application security testing (SAST)**: Static code analysis tools like Checkmarx or Snyk monitor code in the CI/CD pipeline. The build breaks any time there are security vulnerabilities above a certain threshold. Thresholds are set based on service needs.
- ☐ **Dynamic application security testing (DAST) / penetration (pen) testing**: Automated DAST runs at appropriate intervals. Manual DAST or pen testing runs according to the security requirements of the service or company. As a note, some companies require DAST or pen testing prior to large changes or launches. Others run them quarterly. Your production readiness checklist should include the appropriate cadence for your situation.
- ☐ **Dependency scan**: All dependencies are using the latest or patched versions. For this, consider automating the scan using tools like FOSSA or Nexus Vulnerability Scanner to validate versions and licenses.

Governance, Risk, and Compliance (GRC)

- ☐ **GRC documentation**: GRC checklists have been completed as required. Many companies have a separate GRC system available. In that case, this checklist indicates its completion and documentation.

- [ ] **Confidentiality, integrity, availability (CIA) rating**: The CIA rating of the service has been documented and published

Development

- [ ] **Follow best practices from Cloud Provider** (e.g. AWS Aurora) and prepare for Fast Failover
- [ ] **Data Org support** (external team might have such requirements) - DB needs to be in provisioned mode instead of serverless if Bin Logs are used for exporting data, use bigger instances than t.small ones (e.g. t.small instances don’t support IAM access)
- [ ] **Database’s Connection string** and Connection pool configured for the needed workload
- [ ] **Maintenance window defined**
- [ ] **Logging:** All logs are written to STDOUT / STDERR. Logs are written in JSON. Configured verbosity levels. check - https://12factor.net/logs. Do not log any sensitive data.
- [ ] **Integration with monitoring platforms.** Dashboards in place. (e.g. NewRelic / Prometheus / Grafana)
- [ ] **Monitoring dashboards with Business Metrics** (e.g. New Relic / Prometheus / Grafana)
- [ ] **Readme.md** - self-explanatory service name, how to run it locally and domain/subdomain, bounded context described
- [ ] **Architecture docs / C4 Model diagrams**
- [ ] **Service Catalog integration** (e.g. Backstage)
- [ ] API Open Specification file in root directory openapi.yaml
- [ ] **API versioning** if needed