KesatriaAwan

PREPARED FOR UPM iDEC

DEVOPS CONSULTATION · APRIL 2026

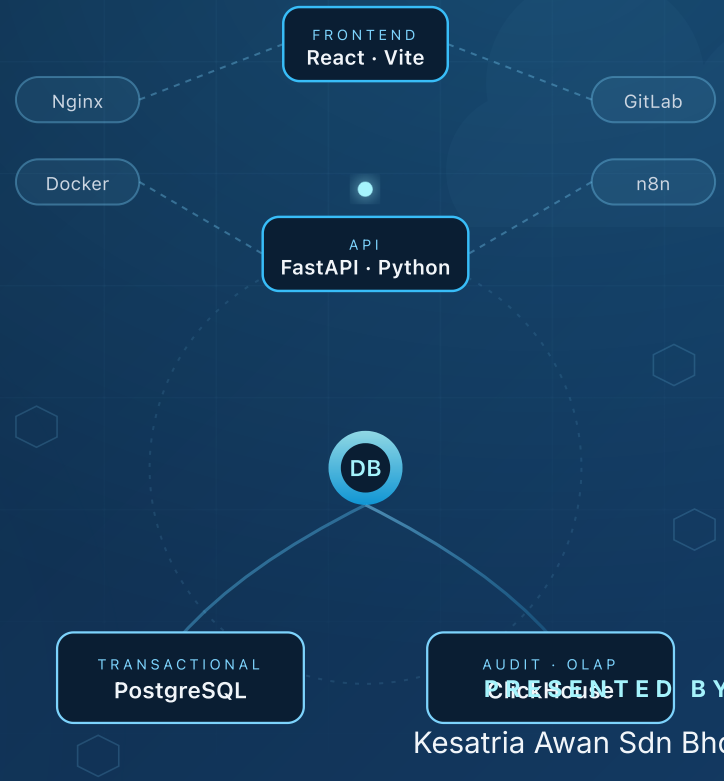
Strengthening DevOps for Putra Portal

A practical conversation with the iDEC Application Development team — covering delivery, testing, security, observability, and the small habits that compound into reliability.

React · FastAPI · Docker · GitLab · PostgreSQL · ClickHouse · n8n · Portainer

MEETING

14 April 2026 · iDEC GAMMA, Putra Infoport, UPM



PRESENTED BY

Kesatria Awan Sdn Bhd

Our goal today

Help Putra Portal ship software **faster**, **safer**, and with **less stress** — starting next week.

No sales pitch. No frameworks you do not need. Just practical next steps.

Kesatria Awan

A Malaysian DevOps & cloud engineering consultancy.
We help teams get from *"it works on our server"* to *"it ships with confidence"*.

What we will cover

1. What we saw in your current setup
2. What is already strong
3. Where we see room to grow
4. What to do this week, this month, this quarter
5. Open discussion

Please interrupt anywhere. This is a discussion, not a lecture.

Part 1

Your current setup

Putra Portal, as we understand it

🖥️

User sees
React + Vite
through Nginx

⚙️

Business logic
FastAPI
Python + raw SQL

🗄️

Data
PostgreSQL — transactions
ClickHouse — audit logs

📦

Packaged as
Docker + Compose
managed via Portainer

🚀

Built & deployed by
GitLab + GitLab Runner

🔁

Integrated with
n8n — workflow automation

Based on the three infographics you shared before the session.

How we read your three infographics

INFOGRAPHIC 1

Seni Bina Digital Putra Portal

What we saw: a clean end-to-end flow — Nginx → React → FastAPI → PostgreSQL, with ClickHouse for audit.

What it tells us: the *architecture* is sound. The *delivery process* around it is where strengthening pays off most.

INFOGRAPHIC 2

Seni Bina Ekosistem DevOps

What we saw: a layered view — access, API, data, and a DevOps lifecycle built around GitLab + Docker + n8n + Portainer.

What it tells us: monitoring & observability are not yet a first-class layer. That is the single biggest visible gap.

INFOGRAPHIC 3

Ekosistem & Alatan DevOps

What we saw: tools for code, CI/CD, containers, automation and access — each with a clear owner.

What it tells us: security scanning and infrastructure-as-code are not yet on the map. Both are low-effort, high-value additions.

Everything that follows is grounded in what you already sent us.

Honest first impression

You have very good bones.

The stack choices are sensible.
The question is not *"what should you replace?"*
It is *"how do we strengthen what you already have?"*

Part 2

What is already working well

Four things you already got right

1. GitLab as your single platform

Code, pipelines, registry — all in one place. One audit trail. Easy to onboard new engineers.

2. Everything in containers

"Works on my laptop" = "works in production". Also makes scaling up to Kubernetes later a small step.

3. Right database for the right job

PostgreSQL for transactions. ClickHouse for audit logs. Audit queries do not starve your user-facing ones.

4. FastAPI & Portainer

A modern, testable backend. A safe visual interface for ops — fewer 2am CLI accidents.

Our recommendations **protect and extend** these choices — not replace them.

Part 3

Where we see room to grow

Seven simple themes. One slide each.

How to read the next seven slides

🤔

The pain

What usually goes wrong at your stage

💡

The shift

The single most important change

🎯

The win

What gets better for your team

Effort & impact rating on each slide — so you can prioritise.

1. Ship smaller, ship more often

🤔 THE PAIN

Changes pile up for weeks. Release day is scary. Rollback is a manual adventure.

💡 THE SHIFT

Merge small changes daily. Use **feature flags** to hide unfinished work. Tag releases; never deploy from a branch.

🎯 THE WIN

Small changes fail small. Releases become boring — and boring is good.

Effort: Low · **Impact:** High · *Mostly a process change.*

2. Make the computer test your code

🤔 THE PAIN

Bugs found by users, not by tests. Manual QA is slow and cannot keep up.

💡 THE SHIFT

Build a test pyramid: many fast unit tests, some API tests against a **real database**, a few end-to-end tests.

🎯 THE WIN

Every merge is verified in minutes. Your raw-SQL code gets the safety net it needs.

Effort: Medium · **Impact:** Very High · *Biggest single lever for reducing production bugs.*

3. Strengthen the delivery pipeline

🤔 THE PAIN

Pipeline only builds and deploys. Problems slip past into production.

💡 THE SHIFT

Every merge runs: **lint → type check → tests → security scan → build → deploy to dev**. Feedback in under 5 minutes.

🎯 THE WIN

Humans catch fewer problems because the pipeline catches them first — when fixing is cheap.

Effort: Medium · **Impact:** High · *Your GitLab already supports every step above.*

4. Catch security problems early

🤔 THE PAIN

Vulnerabilities found during annual pen-test, weeks before go-live. Fixing is expensive and stressful.

💡 THE SHIFT

Turn on GitLab's built-in scanners: **secret leaks, vulnerable dependencies, insecure code, container CVEs**. Results appear inside every merge request.

🎯 THE WIN

Security becomes a daily habit, not a yearly fire drill. Maps cleanly to public-sector security controls.

Effort: Low · **Impact:** High · *Mostly flipping switches you already paid for.*

5. See what is happening in production

🤔 THE PAIN

"The portal is slow today." *Why?* Nobody knows until someone searches logs for hours.

💡 THE SHIFT

Add **metrics** (Prometheus + Grafana) and **request traces** (OpenTelemetry). One Grafana dashboard showing response time, error rate, and database health.

🎯 THE WIN

Problems are spotted in minutes, not hours. You stop arguing — the chart shows the answer.

Effort: Medium · **Impact:** Very High · *Single biggest improvement to incident response time.*

6. Rebuild the server, from git, anytime

🤔 THE PAIN

If your server died today, how long to rebuild it — exactly? A week? Longer? Who knows the steps?

💡 THE SHIFT

Describe your servers in code (Ansible). Fresh VM → working stack in under 30 minutes, with one command.

🎯 THE WIN

Dev, staging, and production are guaranteed identical. Disaster recovery is no longer a rumour.

Effort: Medium · **Impact:** High · *Peace of mind you did not know you were missing.*

7. Treat your data like it is irreplaceable

🤔 THE PAIN

Database migrations run manually. Backups exist but have never been restored. "We hope it works."

💡 THE SHIFT

Migrations in git, run by the pipeline. Automated daily backups. **Quarterly restore drills.** If you cannot restore, you do not have a backup.

🎯 THE WIN

Zero-downtime schema changes. Real disaster recovery, not a document in Confluence.

Effort: Low–Medium · **Impact:** Very High when it matters · *The backup you never tested is not a backup.*

Part 4

What to do, and when

Start with five small wins this week

1. **Turn on GitLab secret scanning.** Three lines in CI. Prevents leaked passwords forever.

2. **Tag every Docker image with the git SHA.** Always know exactly what is running in production.

3. **Add a `/healthz` endpoint.** Docker auto-restarts unhealthy containers. Free reliability.

4. **Write three one-page runbooks** for your top incidents. At 3am, memory fails — the runbook saves you.

5. **Test your backup restore** — just once. If it works, schedule it quarterly. If it does not, you just avoided a disaster.

Each of these takes less than half a day.

The 30 / 60 / 90 day plan

Days 1–30
FOUNDATION

- ✓ Turn on GitLab security scans
- ✓ Separate dev / staging / prod
- ✓ Start tracking 4 simple delivery metrics
- ✓ Verify your backups actually restore

Days 31–60
AUTOMATE

- ✓ Full CI pipeline with tests & scans
- ✓ Database migrations via pipeline
- ✓ Prometheus + Grafana dashboard
- ✓ Ansible playbook: VM → working stack

Days 61–90
MATURE

- ✓ Small changes, shipped daily
- ✓ Automated deploys with rollback
- ✓ Define uptime & latency targets
- ✓ First blameless postmortem

Each month builds on the last. No month is wasted even if you stop there.

Putra Portal in 12 months

A realistic picture of where this roadmap takes you.

🚚 Delivery

- Daily small releases, not monthly big ones
- Every deploy can be rolled back in one click
- Production matches `main`, always

🛡️ Quality & security

- Every merge: tests + scans in under 5 minutes
- Vulnerabilities caught at commit, not at pen-test
- Database migrations reviewed like code

🔭 Operations

- One Grafana dashboard answers "is it healthy?"
- Incidents detected by alerts, not by users
- Backups restored on a schedule, not on faith

👥 People

- Runbooks for the top 10 incidents
- Blameless postmortems as a normal habit
- New engineer productive on day one, not week three

None of this requires replacing the stack you already have.

The four numbers to track

You cannot improve what you do not measure. Start with these — a spreadsheet is enough.

HOW OFTEN

Deployments per week

Are you shipping more, over time?

HOW FAST

Commit → production

Hours? Days? Weeks? Shorter is better.

HOW SAFE

% of deploys that break prod

Elite teams: under 15%.

HOW RESILIENT

Time to recover from incident

Minutes? Hours? Shorter is better.

Published by Google DORA. Validated across thousands of engineering teams.

One page for your leadership

Screenshot this slide. It summarises everything.

WHERE YOU ARE TODAY

Sensible stack, working system. Delivery process is mostly manual. Releases are batched and stressful. Monitoring, security scanning, and disaster recovery are informal.

WHERE YOU ARE GOING

Daily small releases. Automated tests & security scans on every change. Metrics dashboard. Infrastructure rebuilt from git in 30 minutes. Restore-tested backups.

WHAT IT TAKES

~90 days of steady work. No new budget for tools — your GitLab already includes most of what is needed. The investment is engineering time, not licences.

WHAT YOU GAIN

Faster delivery. Fewer late-night incidents. Better security posture. Real DR confidence. Onboarding cut from weeks to days. A team that enjoys the work.

One idea to take away

DevOps is not a product you buy.
It is a set of **habits**.

Pick two from this deck. Start Monday. Review next Friday.
That is how transformations actually happen.

Three ways forward

Pick the level of involvement that fits your team, timeline, and budget.

OPTION 1 · YOU-LED

Training & Self-Service

We teach. You do.

2–3 weeks structured training — workshops, runbooks, recorded sessions. Your team takes it from there.

- ✓ Hands-on workshops
- ✓ Templates & runbooks
- ✓ Async Q&A for 30 days

Best if: your team has time to learn-by-doing.

★ RECOMMENDED

OPTION 2 · WE-LED TOGETHER

Collaborative Service

*We do it **with** you, while training you.*

6–10 weeks side-by-side. We pair-build the pipelines, dashboards, and tests *together* with your engineers — until your team owns it.

- ✓ Pair-build CI/CD & observability
- ✓ Knowledge transfer every sprint
- ✓ Your team finishes capable

Best for Putra Portal: momentum without losing ownership.

OPTION 3 · WE-LED

Full-Service / Managed

We do. You receive.

End-to-end delivery. We design, build, and operate the DevOps stack on your behalf.

- ✓ Full implementation
- ✓ Ongoing operations & monitoring
- ✓ SLA-backed support

Best if: you want speed over upskilling.

No obligation from today. A follow-up call in a week is a good next step regardless of which option appeals.

Discussion

We would like the rest of the session to hear from you.

- 💬 What is the single most painful part of your current delivery process?
- 💬 What would you *like* to do but feel blocked from doing?
- 💬 What does "good" look like for Putra Portal in 12 months?

Terima kasih

Thank you to the iDEC Application Development team for the invitation.
We look forward to continuing the conversation.

KESATRIA AWAN KesatriaAwan