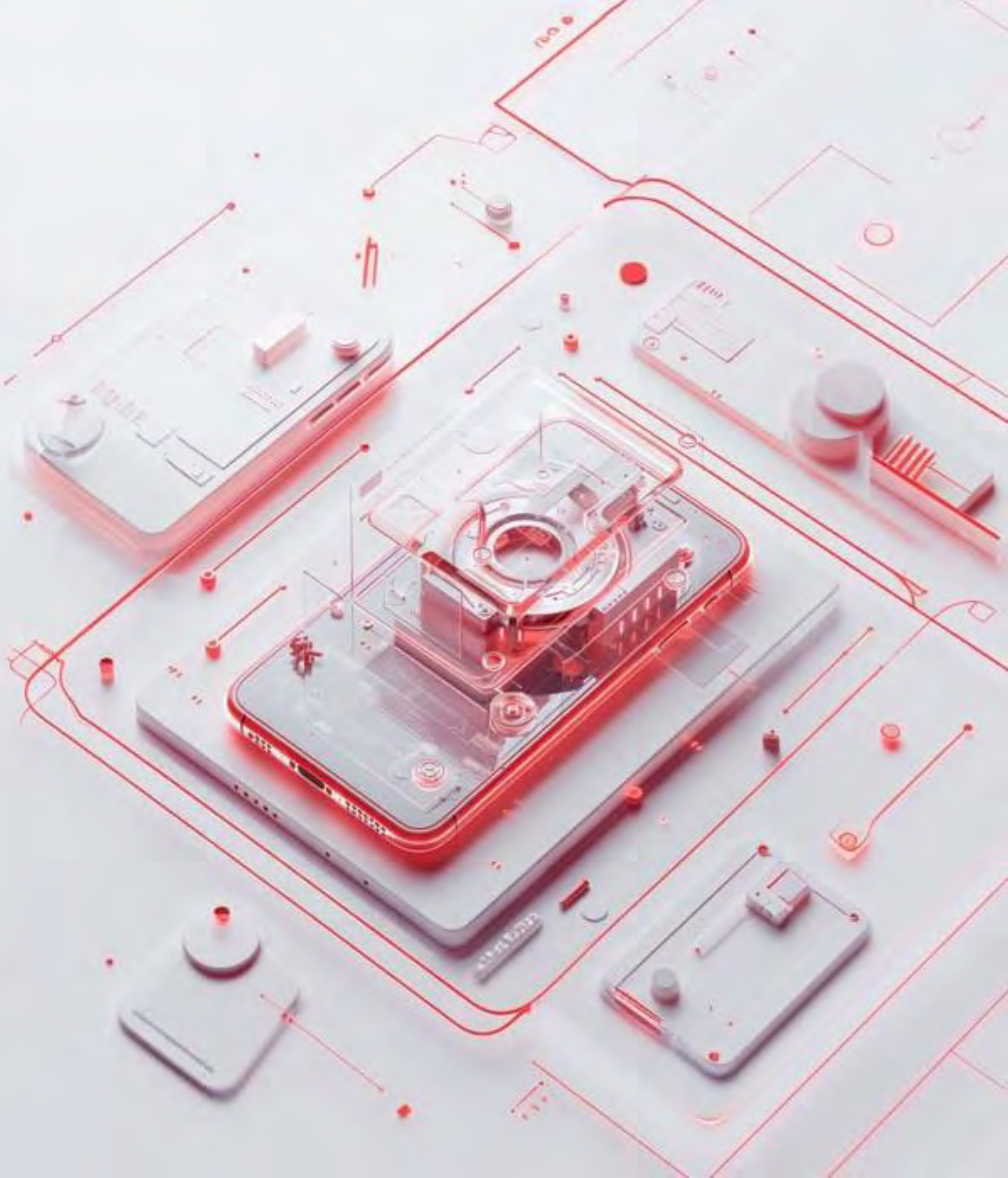
Perelyn

Ultra-Budget AWS: Running Scalable Apps for Pennies

Maksymilian Kałek

Conf42 06.03.2025



Perelyn

About me

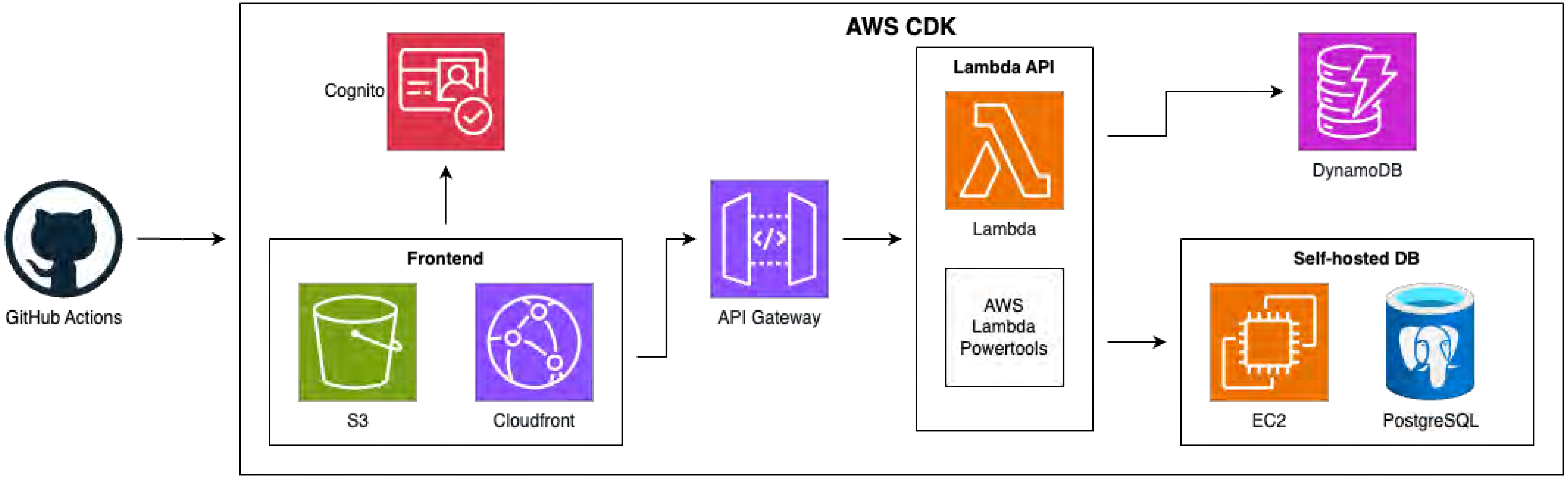
Introduction & Agenda

- Why focus on ultra-budget AWS?
- Key components of the architecture
- Cost-saving techniques
- Challenges & trade-offs
- Real-world examples & cost breakdown

Why Ultra-Budget?

- Who is this for?
  - Indie developers & startups
  - Side projects & MVPs
  - Cost-conscious teams
- Challenges of traditional AWS setups:
  - Overprovisioning leads to high costs
  - Managed services can be expensive at scale
- Solution
  - Strategic use of AWS services to minimize cost while maintaining performance.

Architecture Overview



Frontend: React on S3 + CloudFront + Cognito

- S3
  - Cheap
  - Easy-to-use
  - Scalable

- CloudFront
  - Free HTTPS
  - Global caching
  - Fast access

- Cognito
  - Authentication
  - Session management

Backend: AWS Lambda + Lambda Powertools

Why use AWS Lambda?

- $0 cost when not in use
- Simplicity
- No infrastructure management
- Automatic scaling

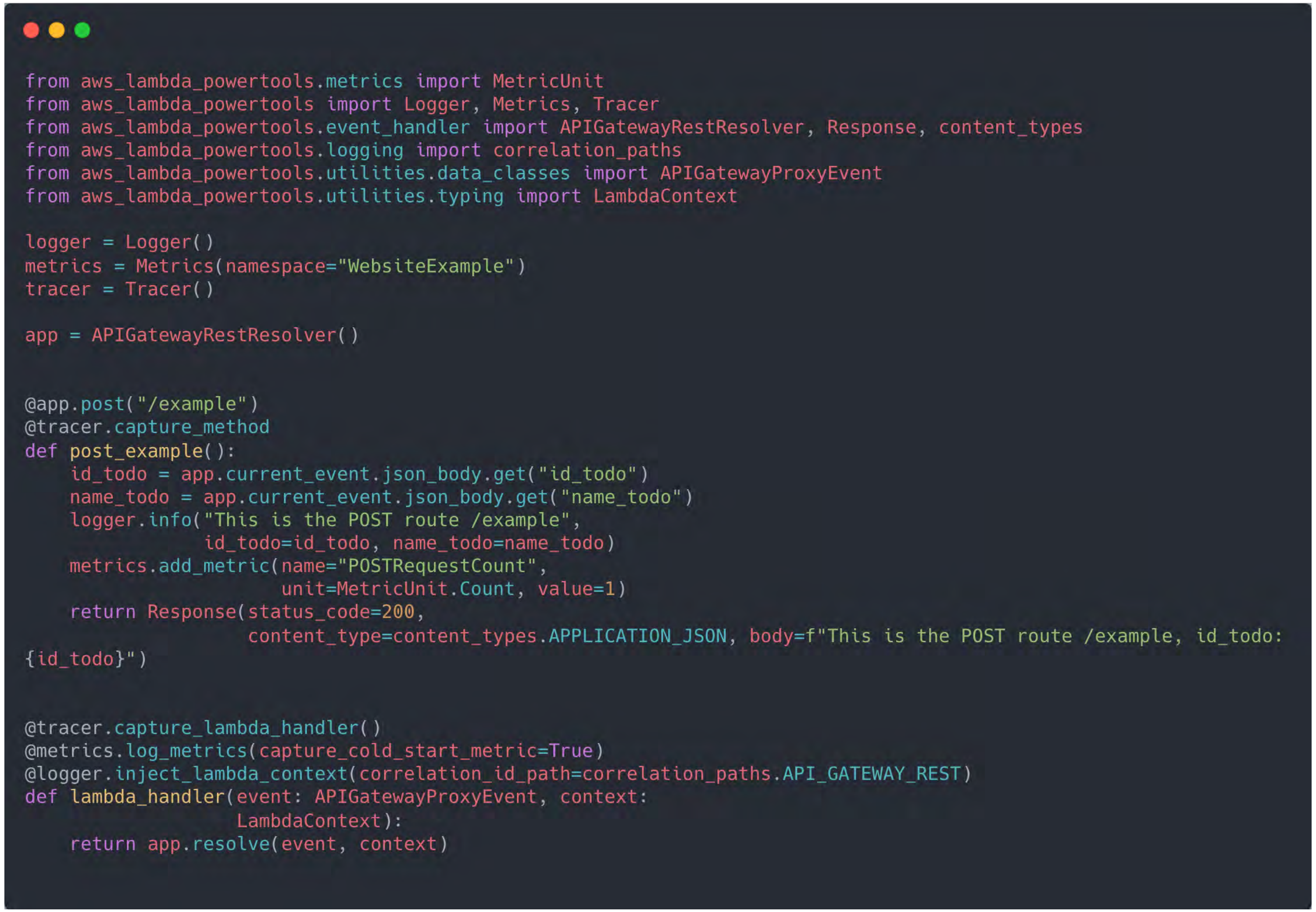
Challenges

- Cold starts
- Stateless execution
- Deployment memory limits

Backend: AWS Lambda + Lambda Powertools

Lambda Powertools

- Event handling similar to common backend frameworks
- Logging and tracing
- Typing
- Parsing
- Validation



```python
from aws_lambda_powertools.metrics import MetricUnit
from aws_lambda_powertools import Logger, Metrics, Tracer
from aws_lambda_powertools.event_handler import APIGatewayRestResolver, Response, content_types
from aws_lambda_powertools.logging import correlation_paths
from aws_lambda_powertools.utilities.data_classes import APIGatewayProxyEvent
from aws_lambda_powertools.utilities.typing import LambdaContext

logger = Logger()
metrics = Metrics(namespace="WebsiteExample")
tracer = Tracer()

app = APIGatewayRestResolver()


@app.post("/example")
@tracer.capture_method
def post_example():
    id_todo = app.current_event.json_body.get("id_todo")
    name_todo = app.current_event.json_body.get("name_todo")
    logger.info("This is the POST route /example",
                id_todo=id_todo, name_todo=name_todo)
    metrics.add_metric(name="POSTRequestCount",
                       unit=MetricUnit.Count, value=1)
    return Response(status_code=200,
                    content_type=content_types.APPLICATION_JSON, body=f"This is the POST route /example, id_todo:
{id_todo}")


@tracer.capture_lambda_handler()
@metrics.log_metrics(capture_cold_start_metric=True)
@logger.inject_lambda_context(correlation_id_path=correlation_paths.API_GATEWAY_REST)
def lambda_handler(event: APIGatewayProxyEvent, context:
                   LambdaContext):
    return app.resolve(event, context)
```

Database: DynamoDB

Why use DynamoDB?

- Fully managed & Scalable
- Pas-as-you-go pricing
- Low latency
- Flexible Schema

Challenges

- Complex Data Modeling
- Limited Query Capabilities

Database (alternative): PosgreSQL on EC2

Why self-host on EC2?

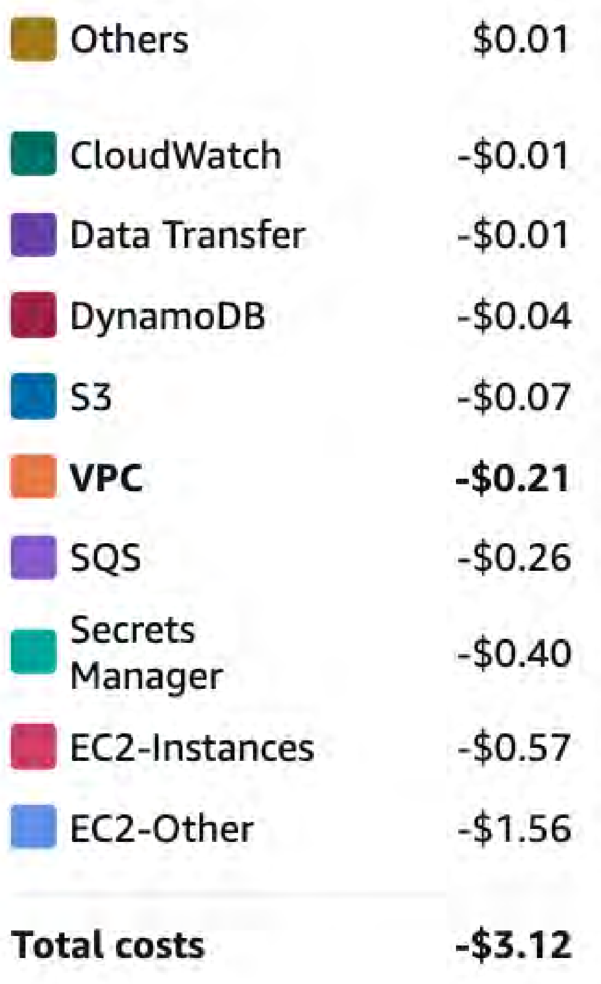
- RDS is costly for small-scale apps
- You need an SQL database for example for vector storage

Challenges

- Maintenance
- Backup management

Cost breakdown & Summary

- Ultra-low total cost
- Running a production-ready system for just a few dollars
- Serverless services (Lambda, API Gateway, CloudFront) = Efficient architecture reducing unnecessary compute expenses
- DynamoDB, S3, and SQS remain budget-friendly – Proving that AWS services can be incredibly cost-efficient when used properly on low traffic



🟫	Others	$0.01
🟩	CloudWatch	-$0.01
🟪	Data Transfer	-$0.01
🟥	DynamoDB	-$0.04
🟦	S3	-$0.07
🟧	**VPC**	**-$0.21**
🟪	SQS	-$0.26
🟩	Secrets Manager	-$0.40
🟥	EC2-Instances	-$0.57
🟦	EC2-Other	-$1.56
	Total costs	**-$3.12**

Thank you!