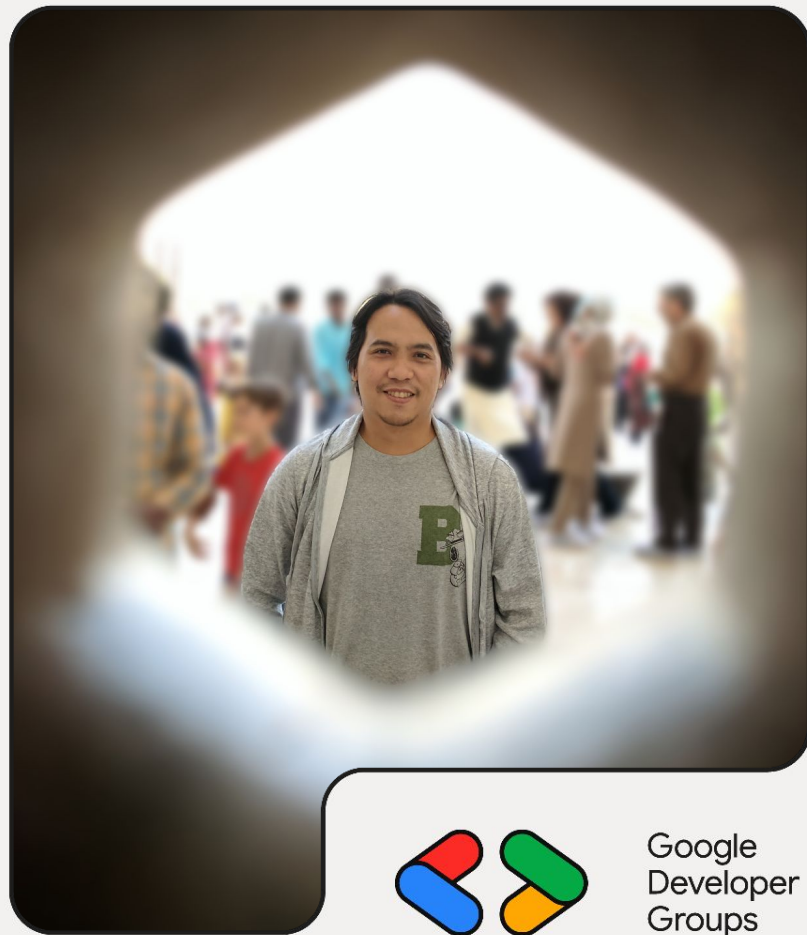


End to End Development with Dart with Serverpod

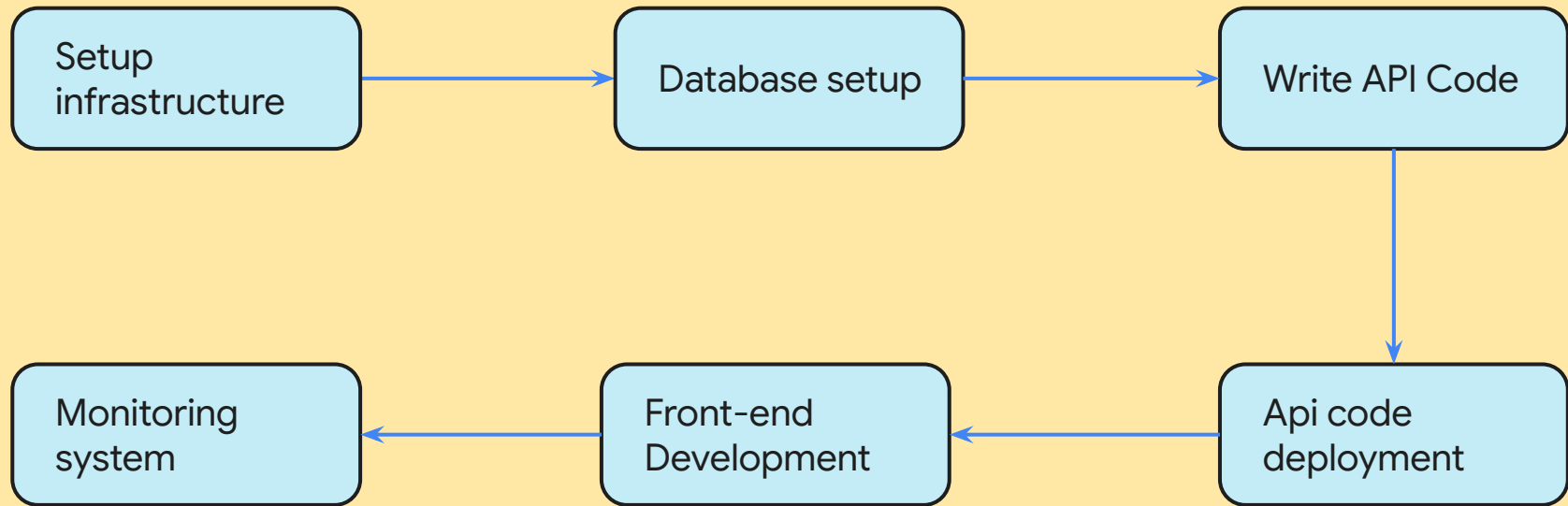
Aldy Chrissandy
Astro AVP, Engineering
Manager - Mobile

Darshan Nair
Deputy Manager,
Bank Islam



Google
Developer
Groups

Development Process



Backend planning

1. API Design

Rest, GraphQL, SOAP, WS, gRPC...

2. Server Setup

Dedicated Server, Serverless, Cloud/SelfHosting, Deployment script...

3. Database

In-Memory, Relational, No-SQL, DB Strategy, ...

4. Security

User Authentication, Data Encryption, Role-Based Control

5. Scalability & Performance

Caching, Logging, Error Handling, Monitoring Tools, Alert & Reporting

Relational Database

(1)

Define Tables

(2)

Table relation, constraint and index

(3)

Query Optimizer

Api Development

(1)

Db Connection

(2)

Data serialization

(3)

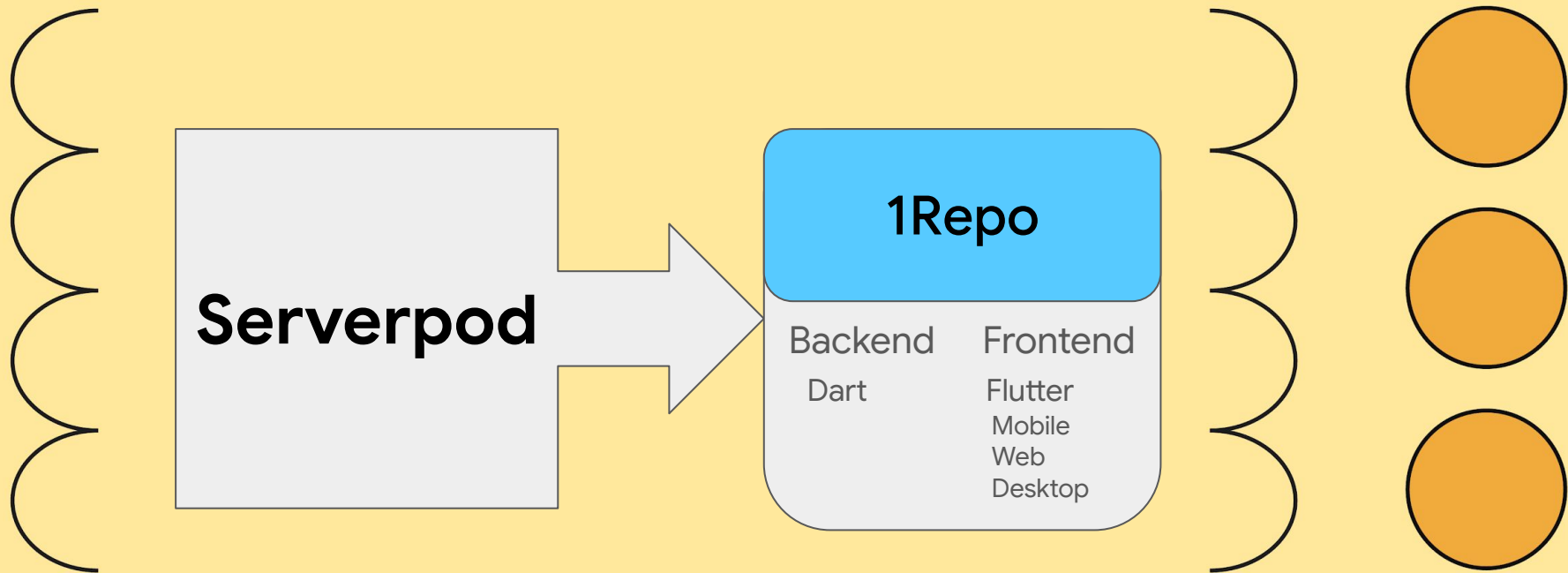
Endpoint and method

(4)

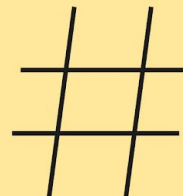
Logging

Frontend Development

- 1 Http Client API calls
- 2 Endpoint and method
- 3 Parsing & Data serialization
- 4 Error handling



Google
Developer
Groups



Mobile
@DevFest

Why Dart + Serverpod

(1)

You are Flutter developer

(2)

Free & Open source

(3)

Accelerate Development time

(4)

Backend Setup for Dummies

(5)

Small development team

Serverpod Features

Built-in Caching

ORM

File Upload

Built-in Auth

Scheduling

Bundle Web
Server

Data Streaming

Health Check

Logging by
default

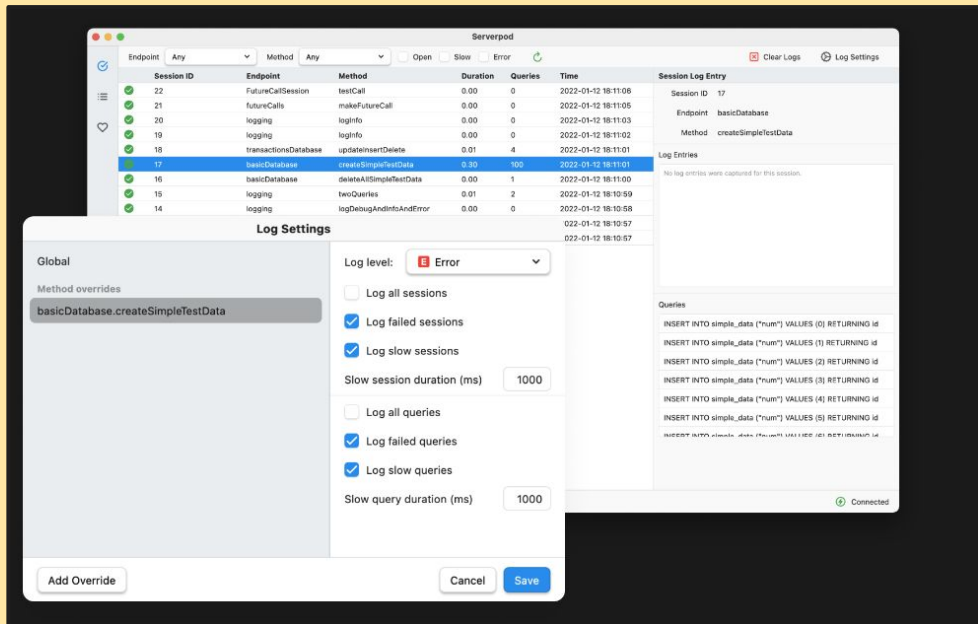
App Insights

Easy Deployment

Custom Module

Serverpod Insight

1. Check error log
2. Monitoring for slow query
3. Checking Server Health



Project Creation

1. Install Serverpod CLI

2. Creating project

3. Start server

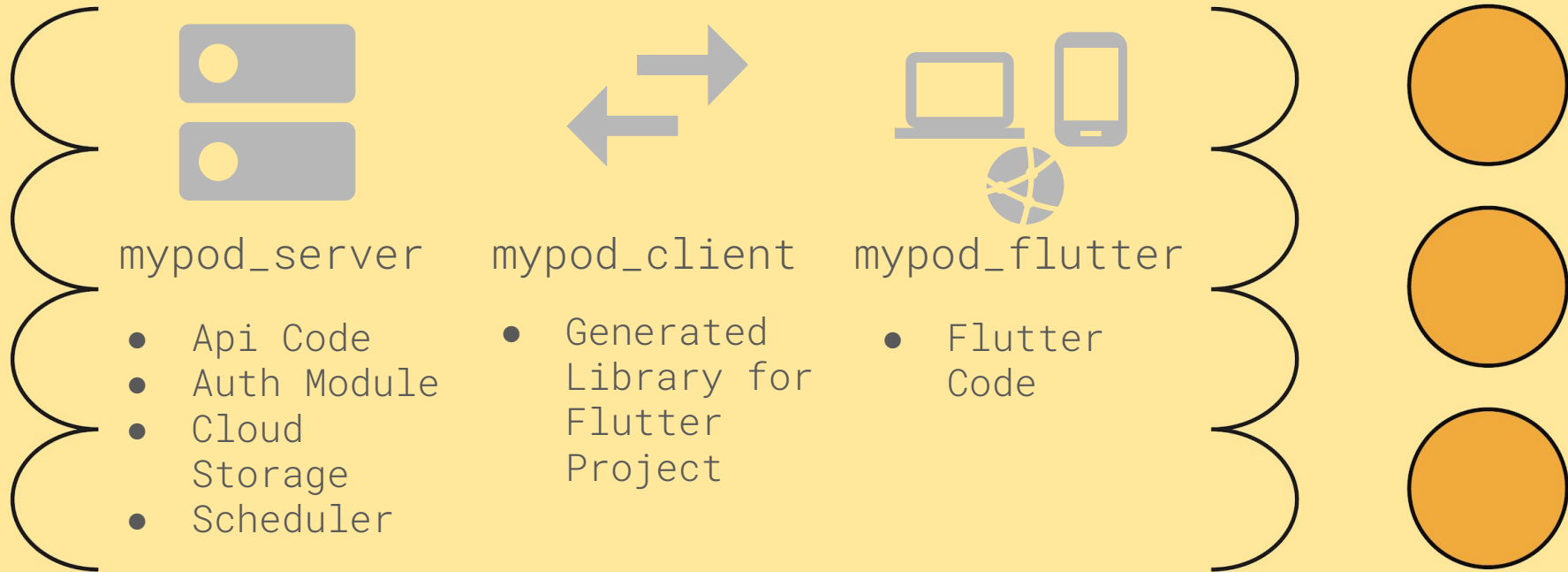
4. Running flutter Apps

```
>>dart pub global activate serverpod_cli
```

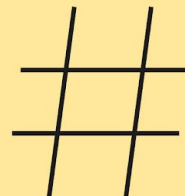
```
>>serverpod create mypod
```

```
>>cd mypod/mypod_server  
>>docker compose up --build --detach  
>>dart bin/main.dart --apply-migrations
```

```
>>cd mypod/mypod_flutter  
>>flutter run -d chrome
```



Google
Developer
Groups



Mobile
@DevFest

Endpoint Creation

1. Place endpoint
2. Extend to Endpoint class
3. Return Future
4. Run
“serverpod generate”

Servercode

```
import 'package:serverpod/serverpod.dart';

class ExampleEndpoint extends Endpoint {

  Future<String> hello(Session session, String name)
  async {
    return 'Hello \'$name\'';
  }
}
```

Flutter code

```
var result = await client.example.hello('World');
```

Data Serialization

1. Models are define in YAML files
2. Place model in models folder
3. Run “serverpod generate”

Data Serialization in YAML

```
class: Company
fields:
  name: String
  foundedDate: DateTime?
  employees: List<Employee>
```

Servercode

```
var myCompany = Company(
  name: 'Serverpod',
  employees: [
    Employee(
      name: 'Viktor',
    ),
  ],
);
```

Flutter code

```
var myCompany = await client.example.getMyCompany();
print(myCompany.name);
```

Database Creation

1. Models are define in YAML files
2. Place model in models folder
3. Define model as table
4. Run “serverpod create-migration”
5. Run “dart run bin/main.dart --apply-migrations”

Data Serialization in YAML

```
# company.yaml
class: Company
table: company
fields:
  name: String
  employees: List<Employee>?, relation

# employee.yaml
class: Employee
table: employee
fields:
  name: String
  address: Address?, relation

# address.yaml
class: Address
table: address
fields:
  addressId: int
  street: String
indexes:
  user_address_unique_idx:
    fields: addressId
    unique: true
```

Db Basic

1. Persisting data
2. Extend to Endpoint class
3. Return Future
4. Run
“serverpod generate”

Servercode

```
class ExampleEndpoint extends Endpoint {  
  Future<void>addCompany(Session s,String name)async {  
    var myCompany = Company(  
      name: name,  
    );  
    var insertedCompany = await  
      Company.db.insertRow(session, myCompany);  
  }  
  
  Future<Company?>findMyCompany(Session s,String name)async {  
    var myCompany = await Company.db.findFirstRow(  
      s,  
      where: (t) => t.name.equals(name),  
    );  
    return myCompany;  
  }}  
}
```

Flutter code

```
await client.example.addCompany('Serverpod');
```

```
var result = await  
client.example.findMyCompany('Serverpod');
```


Caching

1. Define cache key
2. Retrieve object from cache
3. If null get data from The source
4. Put data in cache and Set the lifetime

Servercode

```
class ExampleEndpoint extends Endpoint {  
  
  Future<UserData> getUserData(Session s,int userId) async {  
    var cacheKey = 'UserData-$userId';  
  
    var userData = await  
      session.caches.local.get<UserData>(cacheKey);  
  
    if (userData == null) {  
      userData = UserData.db.findById(session, userId);  
  
      await session.caches.local.put(cacheKey, userData!,  
        lifetime: Duration(minutes: 5)  
      );  
    }  
    return userData;  
  }  
}
```

Scheduling

1. Extend to FutureCall class
2. Define future call name
3. Register Future call name in server main run method
4. Call function anywhere with pass the future call name

Servercode

```
import 'package:serverpod/serverpod.dart';

class ExampleFutureCall extends FutureCall<MyModel> {
  @override
  Future<void> invoke(Session session, MyModel? object) async {
    //Do Something
  }
}

#server.dart
void run(List<String> args) async {
  final pod = Serverpod(
    args,
    Protocol(),
    Endpoints(),
  );

  ...

  pod.registerFutureCall(ExampleFutureCall(), 'exampleFutureCall');
}

#Invoke
await session.serverpod.futureCallAtTime(
  'exampleFutureCall',
  data,
  DateTime(2025, 1, 1),
);
```

Log & Exception

1. Define exception in yaml

Data Serialization in YAML

```
#my_enum_spy.yaml
enum: MyErrorType
Values:
  - BadRequest
  - Unauthorized
  - InternalServerError
```

```
#exception_spy.yaml
exception: MyException
fields:
  message: String
  errorType: MyErrorType
```

Servercode

```
Future<Company?>findMyCompany(Session s,String name)async {
  ...some processing...
  if (failure) {
    final e = MyException(message: 'Failed to do thingy',
      errorType: MyEnum.thingyError,);
    session.log('Some Message', exception: e);
    throw e;
  }
}
```

Flutter code

```
try {
  client.example.doThingy();
} on MyException catch(e) {
  print(e.message);
} catch(e) {
  print('Something else went wrong.');
```

Google Cloud Deployment

1. Deployment tools

2. Setups and Deployment

3. What is deployed

Terraform, Paid Google Cloud Platform, Registered custom domain with Terraform, Project on Github

- Create New GCP Project
- Create service Account
- Enabling APIs
- Setup and verify domain
- Create Docker Artifact Registry
- Deploy Docker Container
- Configure Terraform on GCP
- Configure Serverpod config
- Deploy infrastructure using Terraform

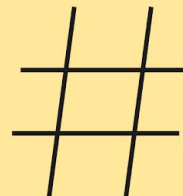
- api.examplepod.com: API server entry point
- app.examplepod.com: Web server
- insights.examplepod.com: Serverpod Insights access
- database.examplepod.com: External database access
- storage.examplepod.com: Public storage access



Thank You



Google
Developer
Groups



Mobile
@DevFest