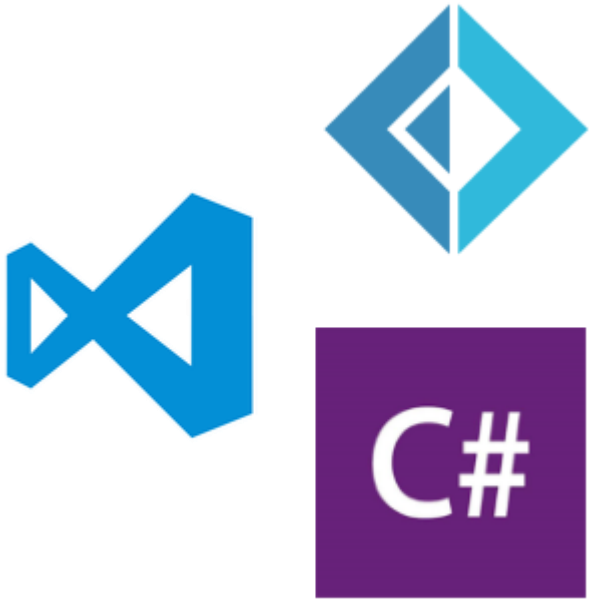
ASP.NET Core приложения под Linux в продакшене

Денис Иванов
denis@ivanovdenis.ru
@denisivanov

Обо мне



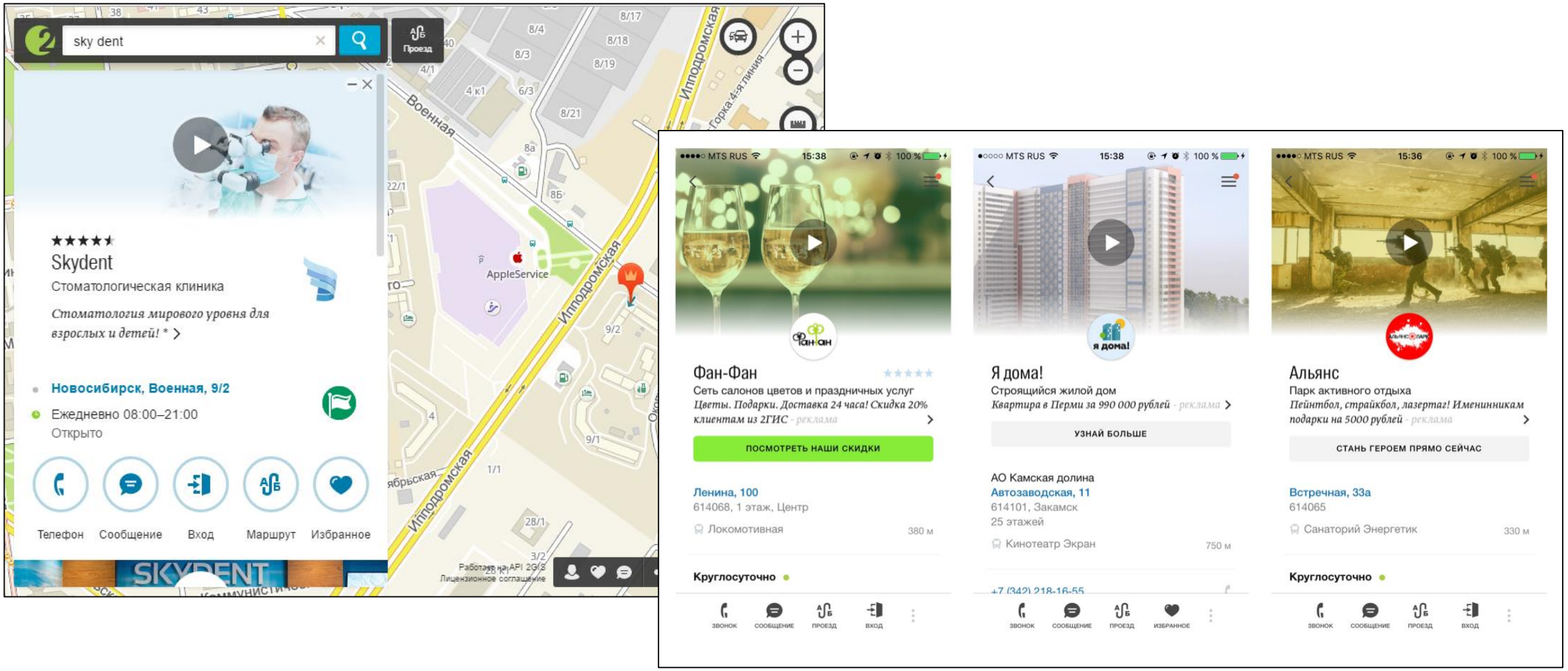
Цель

Поделиться опытом разработки и запуска в продакшен REST-сервисов на ASP.NET Core на Kubernetes

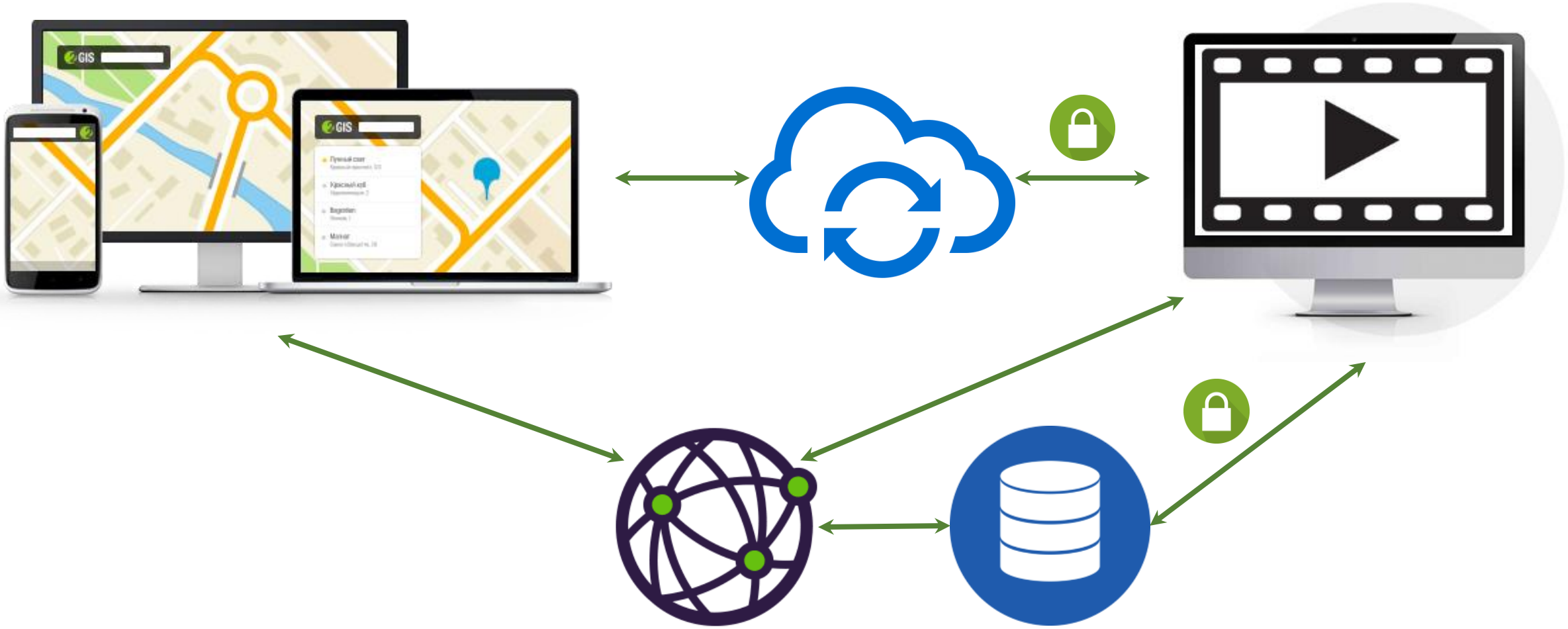
План

- Коротко о сервисе
- On-premise платформа
- .NET Core, ASP.NET Core, must-have библиотеки
- Билд
- Деплой
- Тестирование
- Performance
  - Кэширование
  - Асинхронность и многопоточность

Коротко о сервисе



Сервис видеорекламы



Сервис видеорекламы

- 99.99% доступность по миру
- Время ответа 200ms*

Сервис видеорекламы

Сервис видеорекламы

Почему Linux

- Существующая on-premise платформа
  - GitLab CI
  - CI starting kit на основе make
  - Docker hub & docker images
- Компоненты на любом технологическом стеке
- Kubernetes

The Twelve-Factor App (1-5)

- Одно приложение – один репозиторий
- Зависимости – вместе с приложением
- Конфигурация через окружение
- Используемые сервисы как ресурсы
- Фазы билда, создания образов и исполнения разделены
- Сервисы – отдельные stateless процессы

https://12factor.net

The Twelve-Factor App (6-12)

- Port binding
- Масштабирование через процессы
- Быстрая остановка и запуск процессов
- Среды максимально похожи
- Логирование в stdout
- Административные процессы

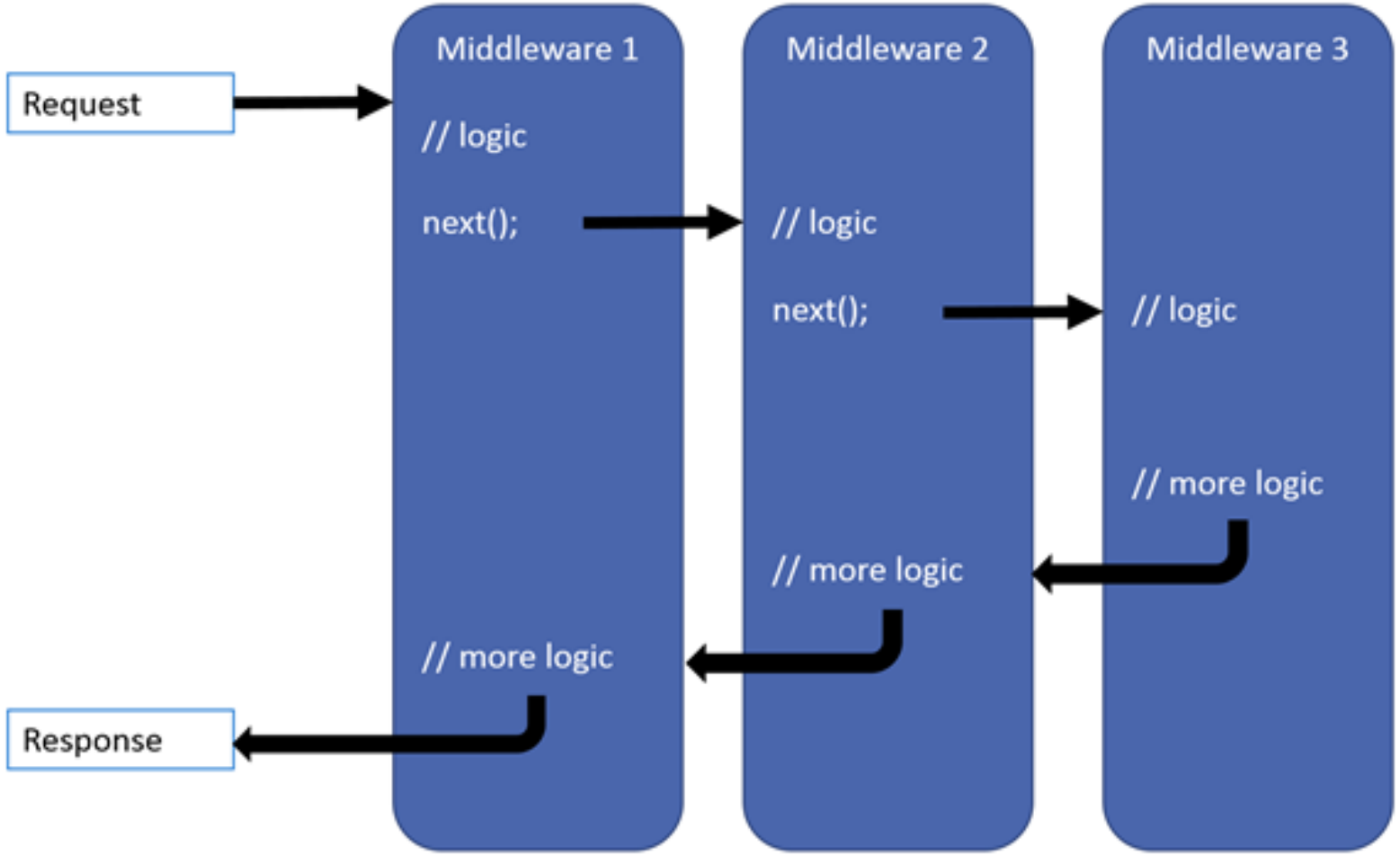
https://12factor.net

- Коротко о сервисе
- On-premise платформа
- **.NET Core, ASP.NET Core, must-have библиотеки**
- Билд
- Деплой
- Тестирование
- Performance
  - Кэширование
  - Асинхронность и многопоточность

.NET Core. Self-contained deployment

- Полный контроль зависимостей
- Явное указание платформы при билде (**win10-x64** / **ubuntu.14.04-x64** / **osx.10.12-x64**)
- Только необходимый фреймворк **netstandard1.6**
  - Microsoft.NETCore.Runtime.CoreCLR
  - Microsoft.NETCore.DotNetHostPolicy

ASP.NET Core


Middleware 1
Middleware 2
Middleware 3
Request
// logic
next();
// logic
next();
// logic
// more logic
// more logic
// more logic
Response

ASP.NET Core middleware

```
public sealed class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        // …
        app.UseExceptionHandler(…);

        app.UseMiddleware<HealthCheckMiddleware>();

        // …
        app.UseMvc();
    }
}
```

```
public sealed class HealthCheckMiddleware
{
    private const string Path = "/healthcheck";
    private readonly RequestDelegate _next;

    public HealthCheckMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        if (!context.Request.Path.Equals(Path, StringComparison.OrdinalIgnoreCase))
        {
            await _next(context);
        }
        else
        {
            context.Response.ContentType = "text/plain";
            context.Response.StatusCode = 200;
            context.Response.Headers.Add(HeaderNames.Connection, "close");
            await context.Response.WriteAsync("OK");
        }
    }
}
```

```
public async Task Invoke(HttpContext context)
{
    if (!context.Request.Path.Equals(
        Path,
        StringComparison.OrdinalIgnoreCase))
    {
        await _next(context);
    }
    else
    {
        context.Response.ContentType = "text/plain";
        context.Response.StatusCode = 200;
        context.Response.Headers.Add(
            HeaderNames.Connection,
            "close");
        await context.Response.WriteAsync("OK");
    }
}
```

ASP.NET Core Logging

```
{
  "dependencies": {
    "Serilog": "2.4.0",
    "Serilog.Enrichers.Thread": "3.0.0",
    "Serilog.Extensions.Logging": "1.4.0",
    "Serilog.Settings.Configuration": "2.2.0",
    "Serilog.Sinks.Console": "2.1.0",
    "Serilog.Sinks.Literate": "2.1.0",
    "Serilog.Formatting.Compact": "1.0.0"
  }
}
```

https://serilog.net/

```
public sealed class Startup
{
    private readonly IConfigurationRoot _configuration;

    public Startup(IHostingEnvironment env)
    {
        var builder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json")
            .AddJsonFile($"appsettings.{env.EnvironmentName}.json");

        _configuration = builder.Build();

        Serilog.Log.Logger = new LoggerConfiguration()
          .ReadFrom.Configuration(_configuration)
          .CreateLogger();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        loggerFactory.AddSerilog();
    }
}
```

appsetting.json

```
{
  "Serilog": {
    "MinimumLevel": "Debug",
    "WriteTo": [
      {
        "Name": "Console",
        "Args": {
           "formatter":
              "Serilog.Formatting.Compact.RenderedCompactJsonFormatter,
               Serilog.Formatting.Compact"
        }
      }
    ],
    "Enrich": [ "FromLogContext", "WithThreadId" ]
  }
}
```

```
public sealed class EnrichSerilogContextMiddleware
{
    private readonly RequestDelegate _next;

    public EnrichSerilogContextMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext context)
    {
        var enrichers = new Stack<ILogEventEnricher>();
        enrichers.Push(new PropertyEnricher("ClientIP", context.Connection.RemoteIpAddress));

        var userAgent = context.Request.Headers[HeaderNames.UserAgent].ToString();
        if (!string.IsNullOrEmpty(userAgent))
        {
            enrichers.Push(new PropertyEnricher("UserAgent", userAgent));
        }

        using (LogContext.PushProperties(enrichers.ToArray()))
        {
          await _next(context);
        }
    }
}
```

```
public async Task Invoke(HttpContext context)
{
    var enrichers = new Stack<ILogEventEnricher>();
    enrichers.Push(new PropertyEnricher("ClientIP",
                    context.Connection.RemoteIpAddress));
    var userAgent =
        context.Request.Headers[HeaderNames.UserAgent]
        .ToString();
    if (!string.IsNullOrEmpty(userAgent))
    {
        enrichers.Push(new PropertyEnricher("UserAgent",
                                            userAgent));
    }
    using (LogContext.PushProperties(enrichers.ToArray()))
    {
      await _next(context);
    }
}
```

ASP.NET API versioning

- https://github.com/Microsoft/aspnet-api-versioning
- Microsoft REST versioning guidelines

- /api/foo?api-version=1.0
- /api/foo?api-version=2.0-Alpha
- /api/foo?api-version=2015-05-01.3.0
- /api/v1/foo
- /api/v2.0-Alpha/foo
- /api/v2015-05-01.3.0/foo

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvcCore(…);
    services.AddApiVersioning(options => options.ReportApiVersions = true);
}

…

[ApiVersion("1.0")]
[Route(“api/medias")]                              // /api/medias
[Route(“api/{version:apiVersion}/medias")]   // /api/1.0/medias
public sealed class MediasController : Controller
{
    [HttpGet("{id}")]
    // /api/medias/id?api-version=1.0 or /api/1.0/medias/id
    public async Task<IActionResult> Get(long id)
    {
        …
    }
}
```

```
[ApiVersion("1.0")]
[ApiVersion(“2.0")]
[Route(“api/medias")]
[Route(“api/{version:apiVersion}/medias")]
public sealed class MediasController : GatewayController
{
    [HttpGet("{id}")]    // /api/medias/id?api-version=1.0 or
/api/1.0/medias/id
    public async Task<IActionResult> Get(long id)
    {
        …
    }

    [MapToApiVersion(“2.0")]
    [HttpGet("{id}")]    // /api/medias/id?api-version=2.0 or
/api/2.0/medias/id
    public async Task<IActionResult> GetV2(long id)
    {
        …
    }
}
```

https://github.com/domaindrivendev/Swashbuckle.AspNetCore

```
services.AddSwaggerGen(
    x =>
    {
        IApiVersionDescriptionProvider provider;
        foreach (var description in provider.ApiVersionDescriptions)
        {
            x.SwaggerDoc(description.GroupName, new Info { … });
        }
    });

…

app.UseSwagger();
app.UseSwaggerUI(
    c =>
    {
         IApiVersionDescriptionProvider provider;
         foreach (var description in provider.ApiVersionDescriptions)
         {
             options.SwaggerEndpoint(
                 $"/swagger/{description.GroupName}/swagger.json",
                 description.GroupName.ToUpperInvariant());
         }
    });
```

.NET Core, ASP.NET Core, must-have библиотеки

Demo

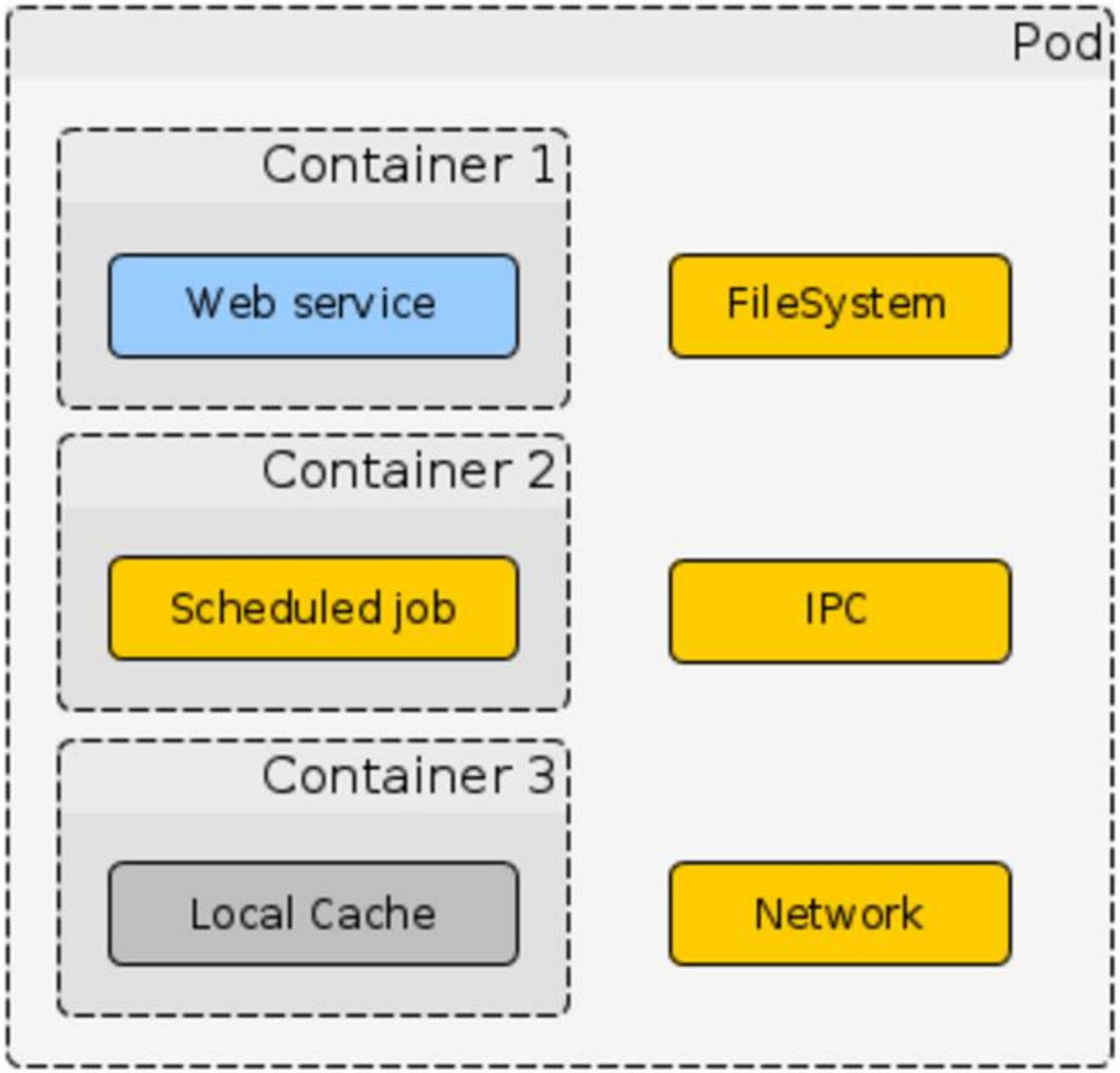
- Коротко о сервисе
- On-premise платформа
- .NET Core, ASP.NET Core, must-have библиотеки
- **Билд**
- Деплой
- Тестирование
- Performance
  - Кэширование
  - Асинхронность и многопоточность

```
build:dotnext-demo:
  image: $REGISTRY/microsoft/aspnetcore-build:1.1.0-
projectjson
  stage: build:app
  script:
    - dotnet restore
    - dotnet publish Demo --configuration Release
     --runtime ubuntu.14.04-x64 --output publish/dotnext-demo
    - dotnet publish UnitTests --configuration Release
     --runtime ubuntu.14.04-x64 --output publish/unit-tests
  tags: [ 2gis, docker ]
  artifacts:
    paths:
      - publish/dotnext-demo/
      - publish/unit-tests/
```

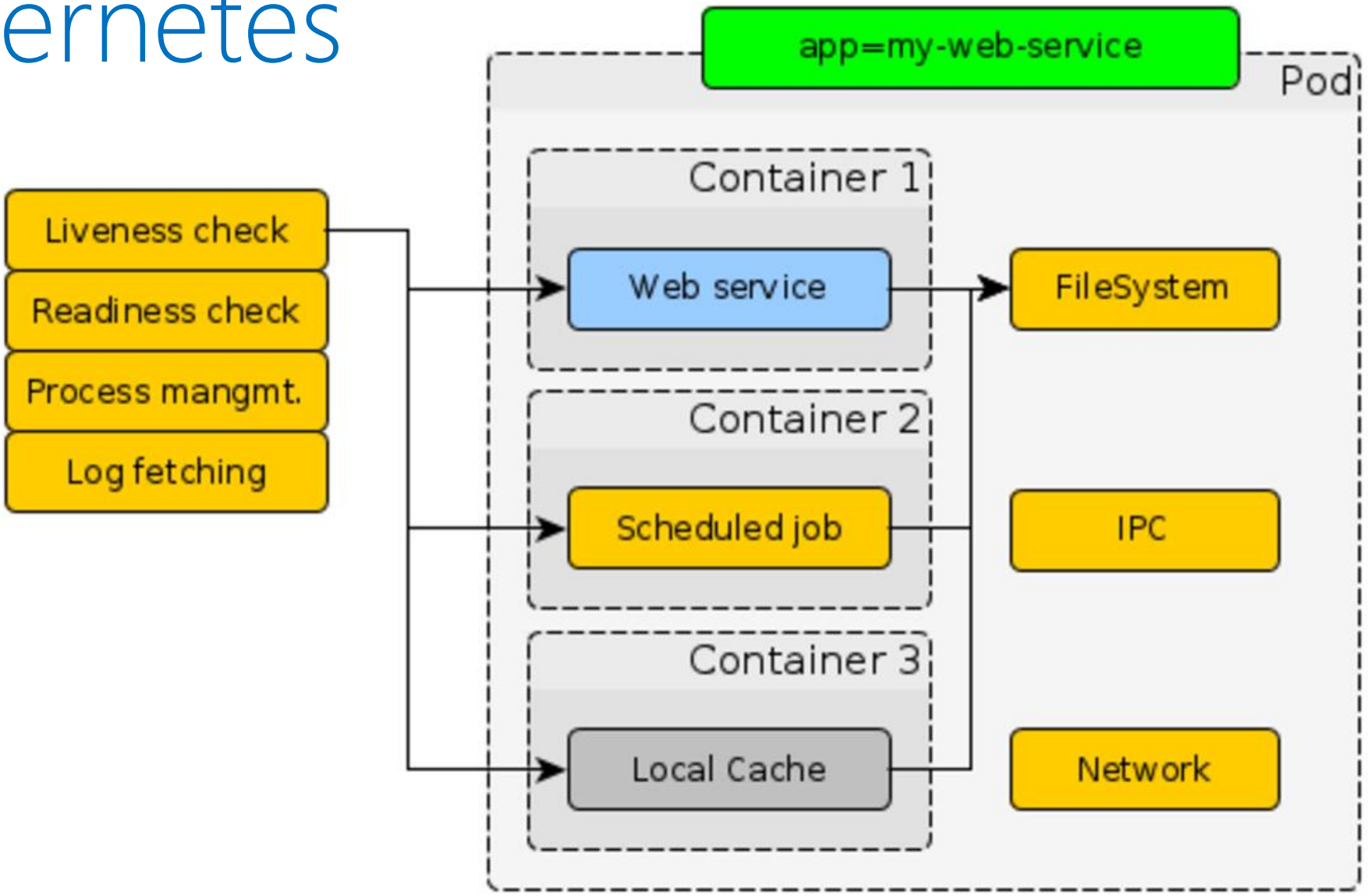
```
build:dotnext-demo-image:
  stage: build:app
  script:
    - IMAGE=my-namespace/dotnext-demo TAG=$CI_TAG
      DOCKER_FILE=publish/dotnext-demo/Dockerfile
      DOCKER_BUILD_CONTEXT=publish/dotnext-demo
      make docker-build
    - IMAGE=my-namespace/dotnext-demo TAG=$CI_TAG
      make docker-push
  tags: [ docker-engine, io ]
  dependencies:
    - build:app
```

- Коротко о сервисе
- On-premise платформа
- .NET Core, ASP.NET Core, must-have библиотеки
- Билд
- **Деплой**
- Тестирование
- Performance
  - Кэширование
  - Асинхронность и многопоточность

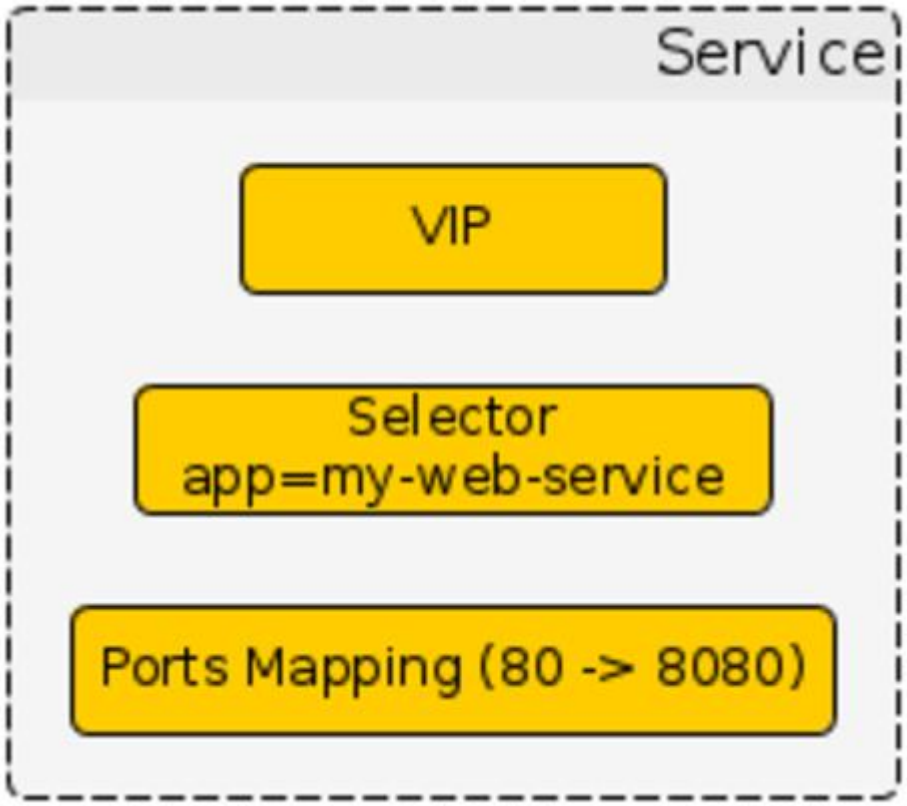
Kubernetes


Pod
Container 1
Web service
FileSystem
Container 2
Scheduled job
IPC
Container 3
Local Cache
Network

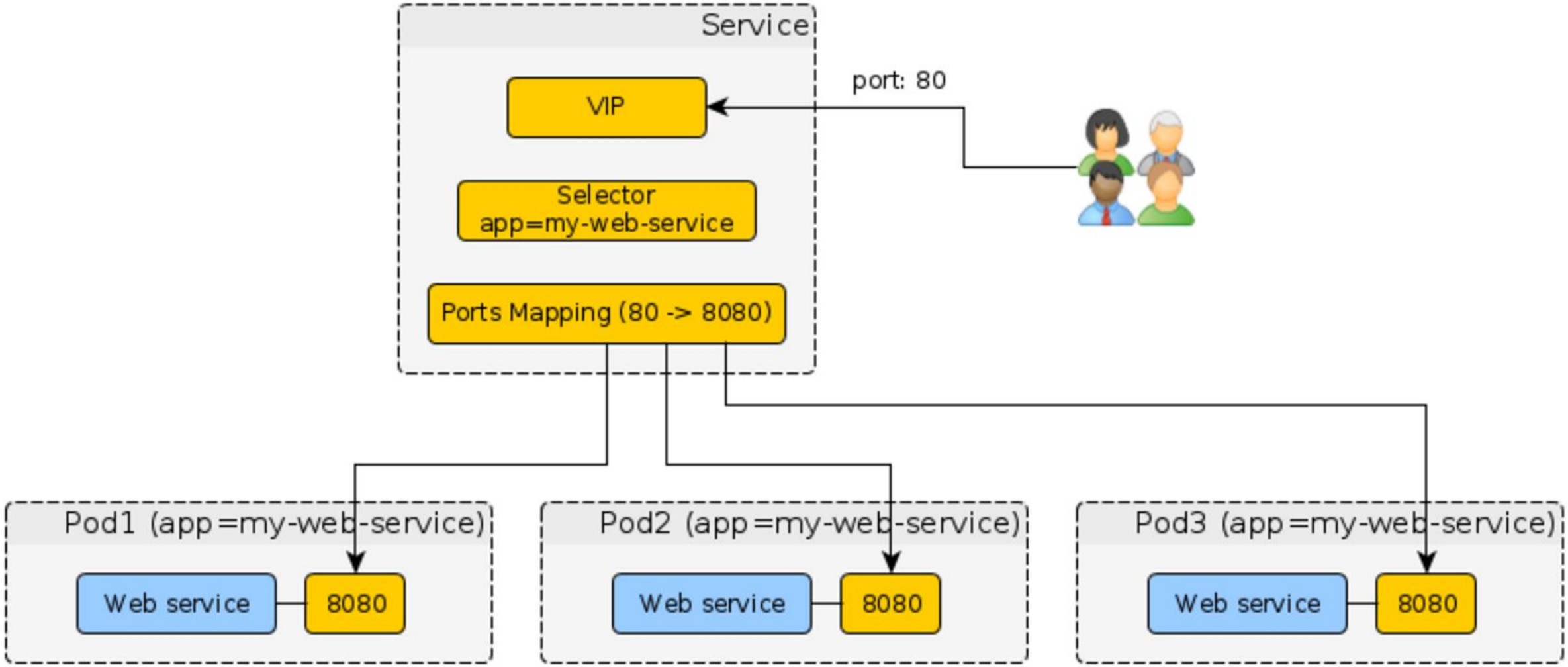
Kubernetes


app=my-web-service
Pod
Container 1
Web service
Container 2
Scheduled job
Container 3
Local Cache
Liveness check
Readiness check
Process mangmt.
Log fetching
FileSystem
IPC
Network

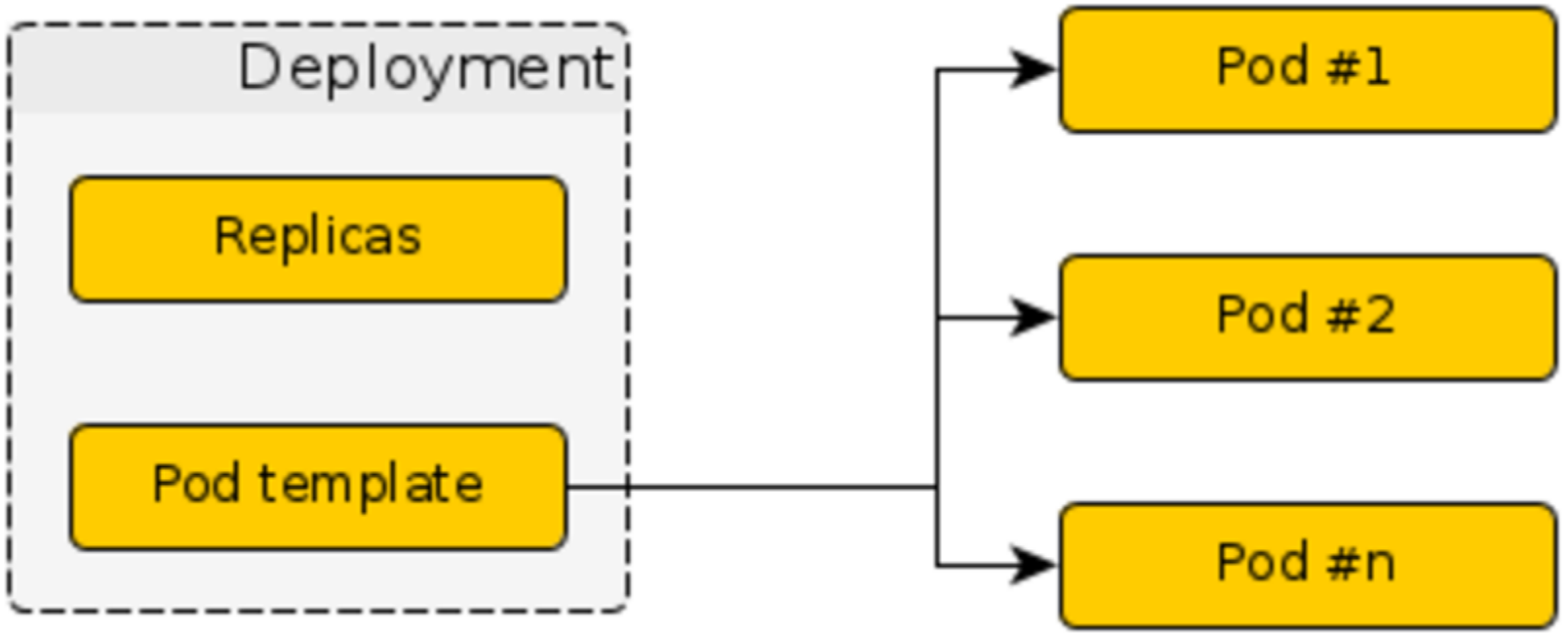
Kubernetes

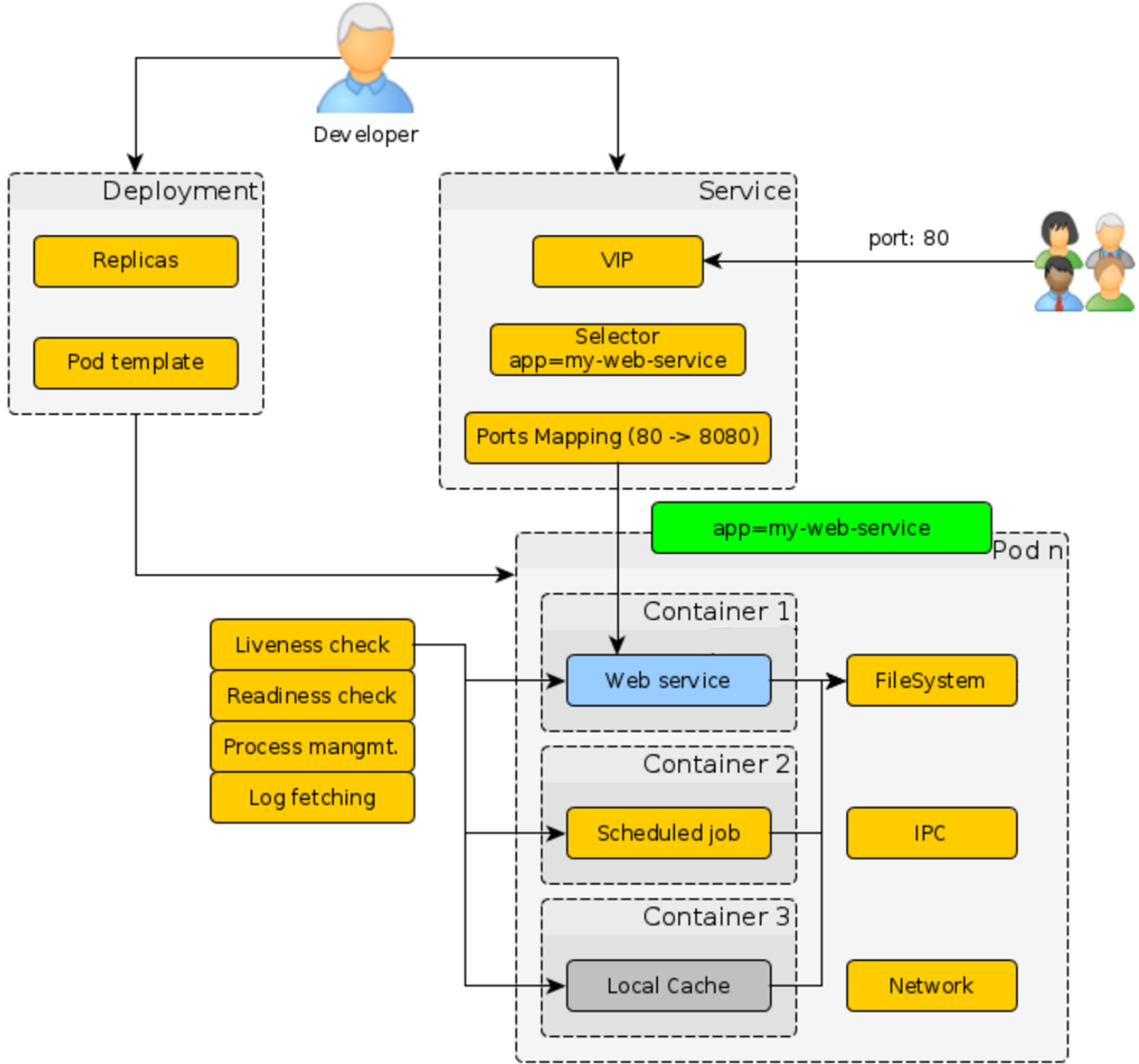

Service
VIP
Selector
app=my-web-service
Ports Mapping (80 -> 8080)

Kubernetes


Service
VIP
port: 80
Selector
app=my-web-service
Ports Mapping (80 -> 8080)
Pod1 (app=my-web-service)
Web service
8080
Pod2 (app=my-web-service)
Web service
8080
Pod3 (app=my-web-service)
Web service
8080

Kubernetes




Developer
Deployment
Replicas
Pod template
Service
VIP
port: 80
Selector
app=my-web-service
Ports Mapping (80 -> 8080)
app=my-web-service
Pod n
Container 1
Web service
FileSystem
Container 2
Scheduled job
IPC
Container 3
Local Cache
Network
Liveness check
Readiness check
Process mangmt.
Log fetching

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: {{ app_name }}
spec:
  replicas: {{ replicas_count }}
  template:
    metadata:
      labels:
        app: {{ app_name }}
    spec:
      containers:
      - name: dotnext
        image: {{ image_path }}:{{ image_version }}
        ports:
        - containerPort: {{ app_port }}
        readinessProbe:
          httpGet: { path: '{{ app_probe_path }}', port: {{ app_port }} }
          initialDelaySeconds: 10
          periodSeconds: 10
        env:
        - name: ASPNETCORE_ENVIRONMENT
          value: {{ env }}
```

```
apiVersion: v1
kind: Service
metadata:
  name: {{ app_name }}
  annotations:
    router.deis.io/domains: "{{ app_name }}"
    router.deis.io/ssl.enforce: "{{ ssl_enforce | default('False') }}"
spec:
  ports:
    - name: http
      port: 80
      targetPort: {{ app_port }}
  selector:
    app: {{ app_name }}
```

```yaml
common:
  replicas_count: 1
  max_unavailable: 0

  k8s_master_uri: https://master.staging.dc-nsk1.hw:6443
  k8s_token: "{{ env='K8S_TOKEN_STAGE' }}"
  k8s_ca_base64: "{{ env='K8S_CA' }}"
  k8s_namespace: my-namespace
  ssl_enforce: true

  app_port: 5000
  app_probe_path: /healthcheck

  image_version: "{{ env='CI_TAG' }}"
  image_path: docker-hub.2gis.ru/my-namespace/dotnext-demo
  env: Stage

dotnext-demo:
  app_name: "dotnext-demo"

  app_limits_cpu: 500m
  app_requests_cpu: 100m
  app_limits_memory: 800Mi
  app_requests_memory: 300Mi

  kubectl:
  - template: deployment.yaml.j2
  - template: service-stage.yaml.j2
```

```yaml
deploy:dotnext-demo-stage:
  stage: deploy:stage
  when: manual
  image: $REGISTRY/2gis-io/k8s-handle:latest
  script:
    - export ENVIRONMENT=Stage
    - k8s-handle deploy
      --config config-stage.yaml
      --section dotnext-demo --sync-mode True
  only:
    - tags
  tags: [ 2gis, docker ]
```

- Коротко о сервисе
- On-premise платформа
- .NET Core, ASP.NET Core, must-have библиотеки
- Билд
- Деплой
- **Тестирование**
- Performance
  - Кэширование
  - Асинхронность и многопоточность

Тестирование

```
build:run-tests:
  image: $REGISTRY/microsoft/dotnet:1.1.1-sdk-1.0.0-preview2-
1-003177
  stage: build:run-tests
  when: always
  script:
    - dotnet restore
    - dotnet test --output publish/unit-tests
      --configuration Release --no-build
  tags: [ 2gis, docker ]
  dependencies:
    - build:app
```

```
.perf:gateway_template: &perf_gateway_template
  stage: test:perf
  environment: perf
  only:
    - master
    - /^perf.*$/
  variables:
    PERF_TEST_PATH: "tests/perf"
    PERF_ARTIFACTS: "target/gatling"
    PERF_GRAPHITE_HOST: "graphite-exporter.perf.os-n3.hw"
    PERF_GRAPHITE_ROOT_PATH_PREFIX: "gatling.service-prefix"
  image: $REGISTRY/perf/tools:1
  artifacts:
    name: perf-reports
    when: always
    expire_in: 7 day
    paths:
      - ${PERF_TEST_PATH}/${PERF_ARTIFACTS}/*
  tags: [perf-n3-1]
```

```
perf:run-tests:
  <<: *perf_gateway_template
  script:
    - export PERF_GRAPHITE_ROOT_PATH_PREFIX
PERF_GRAPHITE_HOST
    - export PERF_APP_HOST=http://${APP_PERF}.web-
staging.2gis.ru
    - cd ${PERF_TEST_PATH}
    - ./run_test.sh --capacity
    - ./run_test.sh --resp_time
  after_script:
    - perfberry-cli logs upload --dir
${PERF_TEST_PATH}/${PERF_ARTIFACTS} --env ${APP_PERF}.web-
staging.2gis.ru gatling ${PERFBERRY_PROJECT_ID}
```

Нагрузочное тестирование

- Лимиты по памяти и процессору
  - 384Mb и 1,5 CPU
- Синхронный (capacity) тест
  - ~24 RPS
- Асинхронный (load) тест
  - Прогрев от 1 до 20 RPS в течение 30 секунд
  - 20 RPS в течение 2 минут

- Коротко о сервисе
- On-premise платформа
- .NET Core, ASP.NET Core, must-have библиотеки
- Билд
- Деплой
- Тестирование
- **Performance**
  - Кэширование
  - Асинхронность и многопоточность

Кеширование

```
[AllowAnonymous]
[HttpGet("{id}")]
[ResponseCache(
    VaryByQueryKeys = new[] { "api-version" },
    Duration = 3600)]
public async Task<IActionResult> Get(long id)
{
    …
}
```

Кеширование

```
[ResponseCache(
    VaryByQueryKeys = new[] { "api-version" },
    Duration = 3600)]
```

- На клиенте
  - Cache-Control header (HTTP 1.1 Caching)
- На сервере
  - Response Caching Middleware (docs)

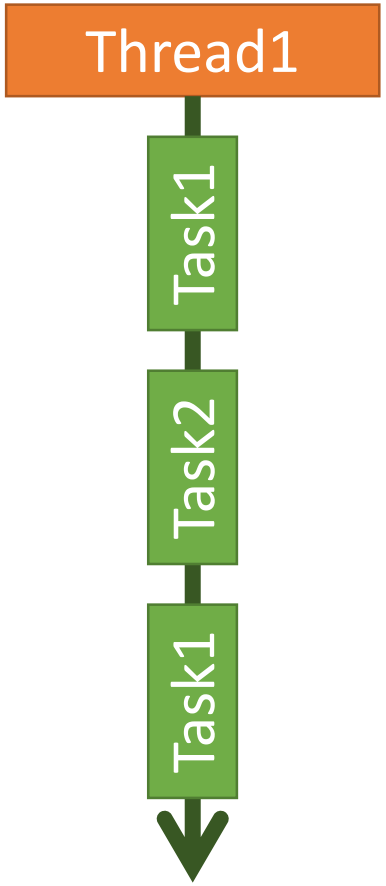
Performance

Асинхронность

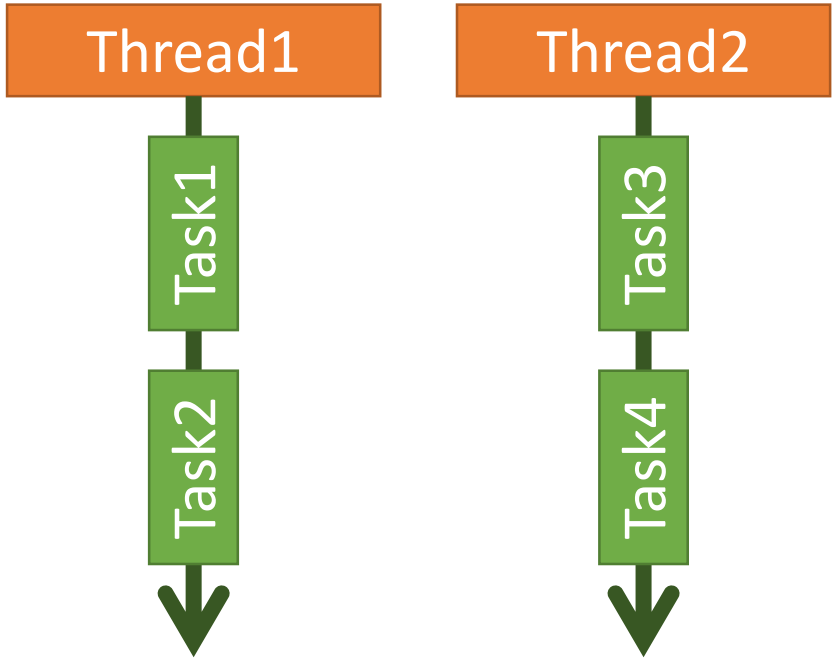
Многопоточность

Performance

Асинхронность



Многопоточность



```
var data =
 await _remoteService.IOBoundOperationAsync(timeoutInSec: 1);
var result = new string[data.Count];

Parallel.ForEach(
 data,
 async (item, loopState, index) =>
  {
   var response =
    await _remoteService.IOBoundOperationAsync(timeoutInSec: 5);
   foreach (var x in response)
   {
    result[index] = x;
   }
  });
```

```
var data =
 await _remoteService.IOBoundOperationAsync(timeoutInSec: 1);
var result = new string[data.Count];

Parallel.ForEach(
 data,
 (item, loopState, index) =>
  {
   var response =
    _remoteService.IOBoundOperationAsync(timeoutInSec: 5);
   foreach (var x in response.Result)
   {
    result[index] = x;
   }
  });
```

```
var data =
 await _remoteService.IOBoundOperationAsync(timeoutInSec: 1);
var result = new string[data.Count];

var tasks = data.Select(
 async (item, index) =>
  {
   var response =
    await _remoteService.IOBoundOperationAsync(timeoutInSec: 5);
   foreach (var x in response)
   {
    result[index] = x;
   }
  });

await Task.WhenAll(tasks);
```

Performance

Demo

Вместо заключения

Вместо заключения

- Не бойтесь использовать .NET Core в продакшене

Вместо заключения

- Не бойтесь использовать .NET Core в продакшене
- Не бойтесь использовать Linux и .NET Core

Вместо заключения

- Не бойтесь использовать .NET Core в продакшене
- Не бойтесь использовать Linux и .NET Core
- Docker и Kubernetes сильно упрощают жизнь

Вместо заключения

- Не бойтесь использовать .NET Core в продакшене
- Не бойтесь использовать Linux и .NET Core
- Docker и Kubernetes сильно упрощают жизнь
- Пишите эффективный код правильно

Спасибо!

https://github.com/denisivan0v/dotnext-piter-2017

Вопросы?

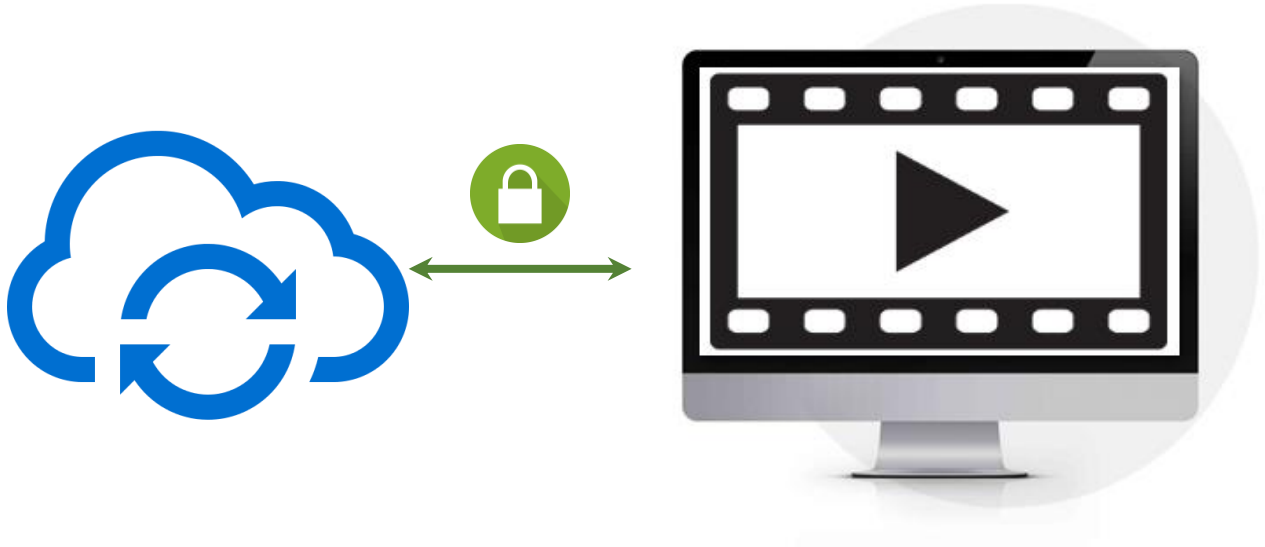
Денис Иванов

@denisivanov

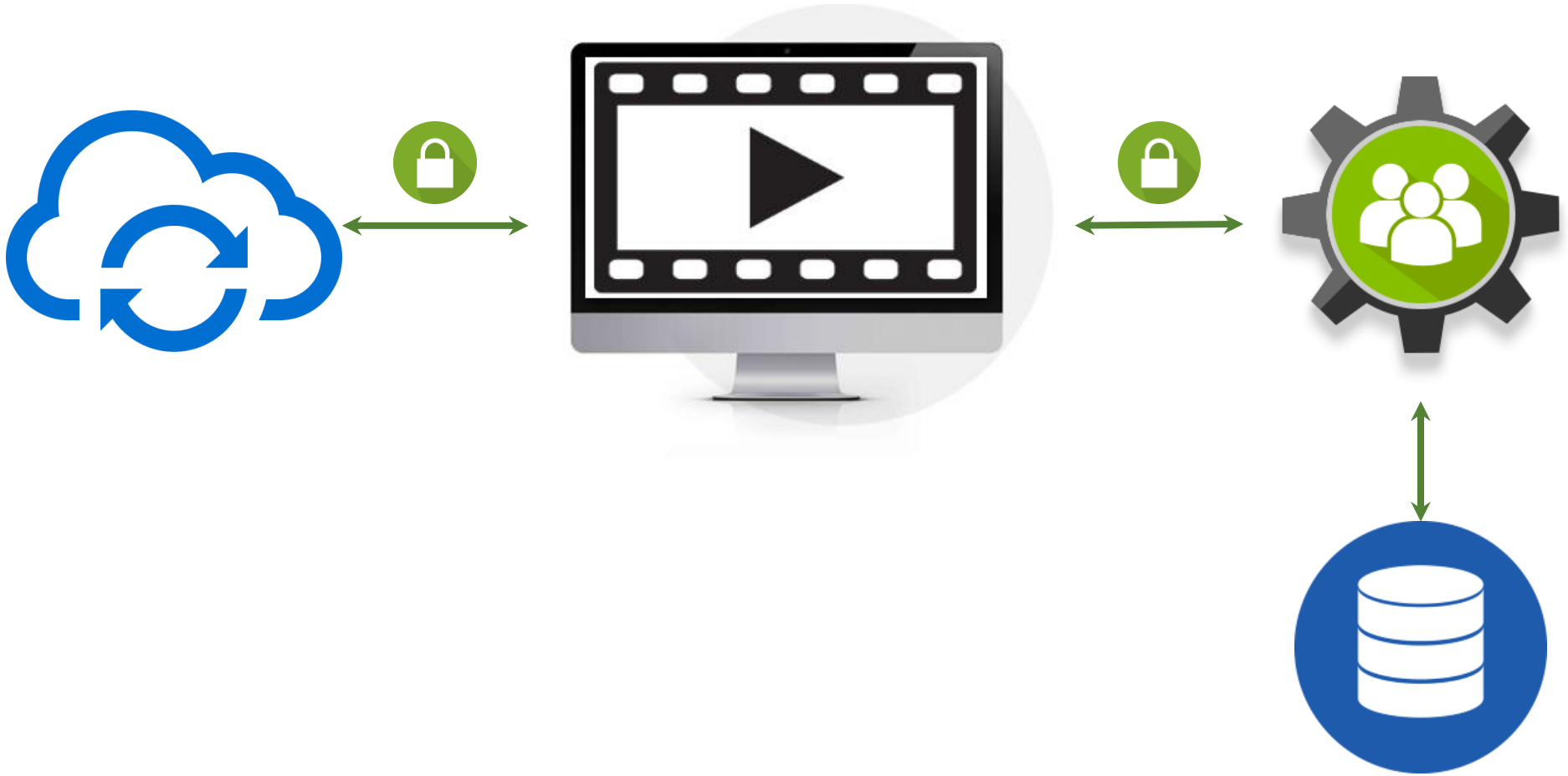
denis@ivanovdenis.ru

https://github.com/denisivan0v

Authentication & authorization



Authentication & authorization



Authentication & authorization

```
var jwtOptions = app.ApplicationServices.GetRequiredService<IOptions<JwtOptions>>();
app.UseJwtBearerAuthentication(
   new JwtBearerOptions
      {
         AutomaticAuthenticate = true,
         AutomaticChallenge = true,
         TokenValidationParameters =
            new TokenValidationParameters
                {
                   ValidateIssuer = true,
                   ValidIssuer = jwtOptions.Value.Issuer,

                   ValidateAudience = true,
                   ValidAudience = jwtOptions.Value.Audience,

                   ValidateIssuerSigningKey = true,
                   IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(jwtOptions.Value.SecretKey)),

                   ValidateLifetime = true,
                   LifetimeValidator = (notBefore, expires, securityToken, validationParameters) =>
                                       {
                                          var utcNow = DateTime.UtcNow;
                                          return notBefore <= utcNow && utcNow <= expires;
                                       }
                }
      });
```

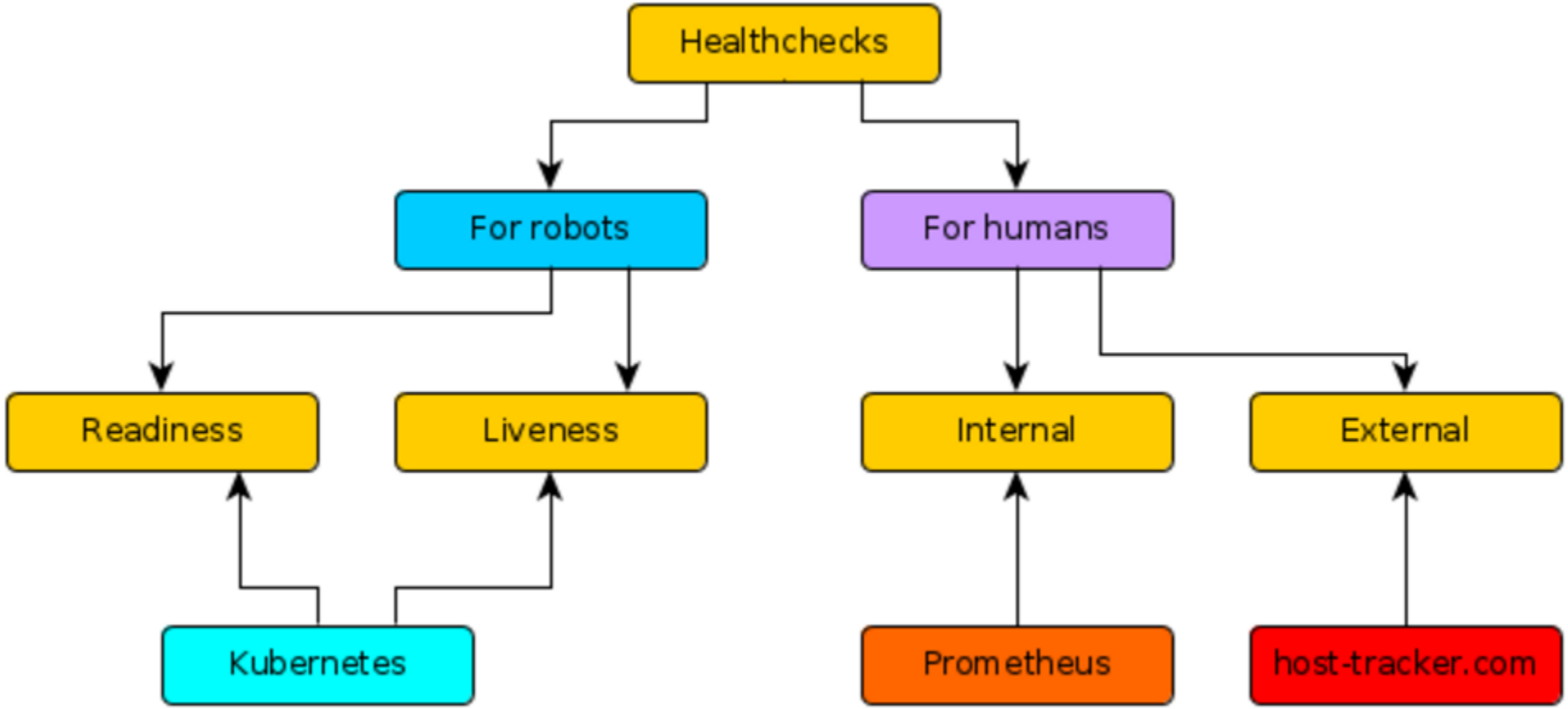
Тестирование

```
import Scenario._
import io.gatling.core.Predef._
import scala.concurrent.duration._
import scala.language.postfixOps

class LoadTest extends Simulation {
  val asserts = Seq(
    global.requestsPerSec.gte(17),
    global.failedRequests.count.is(0),
    details("Get").responseTime.percentile3.lte(700)
  )

  val injectionSteps = Seq(
    rampUsersPerSec(1) to 20 during (30 seconds),
    constantUsersPerSec(20) during (120 seconds)
  )

  setUp(scn().inject(injectionSteps).protocols(httpConf))
    .maxDuration(180 seconds)
    .assertions(asserts)
}
```

Performance

- Асинхронность
- Многопоточность

Эксплуатация



Эксплуатация

- Prometheus server (/metrics)