Zašto nam treba PaaS u Srcu?

Denis Kranjčec, Srce

Zašto Cloud?

Zašto PaaS?

Što naši korisnici očekuju od nas?

- Brz razvoj/inovacije
- Visoku dostupnost/pouzdanost
- Visoke performanse
- Dostupnost usluge bez obzira na lokaciju i korišteni uređaj

Brzina

- Koliko je vremena potrebno da se realizira okolina za novu aplikaciju i da se ta aplikacija pusti u produkciju?
- Koliko je vremena potrebno da se nova verzija aplikacije pusti u produkciju?
- Koliko je vremena potrebno da se ispravi pogreška?
- Koliki je rizik provesti neki eksperiment u produkciji?
- Koliko je vremena potrebno da promjena od samo jedne linije koda pusti u produkciju?

Sigurnost

- Kako onda osigurati stabilnost i dostupnost (ako možemo raditi vrlo brze promjene)?

- Vidljivost
  - moramo osigurati (kroz korištenu arhitekturu i alate) da vidimo problem kada se dogodi. Trebamo pratiti „sve" i pratiti odnos prema „normalnom stanju" (detaljne metrike, nadzor, sustavi upozorenja, vizualizacije).
- Izolacija pogrešaka
  - kako bi se smanjila razina rizika vezanih za pogreške u aplikacijama potrebno je ograničiti broj komponenti ili funkcionalnosti na koje će pojedina pogreška utjecati.
- Otpornost na pogreške
  - potrebno je osigurati da se jedna pogreška ne može kaskadno proširiti kroz sustav („osigurači").
- Automatski oporavak od pogrešaka
  - sa svime navedenim imamo alat za detekciju pogreške, oporavak od pogreške te pružanje (nešto manje) funkcionalnosti klijentima. Primjeri su situacije kada aplikacija nije dostupna ili kada aplikacija generira veliki broj pogrešaka.

Skalabilnost

- Kako osigurati povećanje kapaciteta naše usluge tijekom „sezonskih” povećanja broja zahtjeva/korisnika?

- Virtualnim serverima?
  - Kako osigurati brzo kreiranje i uništavanje novih instanci aplikacije?
  - Kako osigurati da to korisnici ne osjete?

Kako možemo povećati brzinu razvoja, razinu inovacije i razinu kvalitete?

Kako ćemo korisnicima isporučiti aplikaciju nakon što je kod napisan?

CI - Continuous Integration

- Uvođenjem *continuous integration (CI)* prakse razvoja softvera.
- Ključne prakse CI-ja su:

1. Koristiti jedan repozitorij koda.
2. Automatizirati *build. (build - „A set of activities performed to generate, test, inspect and deploy software”)*
3. *Build* treba sadržavati automatsko testiranje koda (*unit test*).
4. Svi programeri (barem) na dnevnoj bazi *push-aju* promjene koda u „centralni” repozitorij koda.
5. Svaki *push* mora pokrenuti automatizirani *build* kompletnog koda iz „centralnog” repozitorija.
6. Neuspjeli *build* se odmah treba riješiti/ispraviti.
7. Izrada *build-a* mora biti brza.
8. Aplikacija se mora testirati u testnom okruženju koje je klon produkcijskog okruženja
9. Svima mora biti lako dostupan rezultat zadnjeg *build-a* (.exe, .jar, .war, …).
10. Svi mogu jednostavno vidjeti što se događa – kakvo je stanje zadnjeg/pojedinog *build-a*.

Kako ćemo odraditi *build, deploy, test and release* proces?

- Aplikaciju ćemo pokrenuti na svom računalu i ručno testirati novu funkcionalnost?
- Obavit ćemo svih XY koraka iz detaljne dokumentacije o tome kako se aplikacija pušta u produkciju?
- Spojit ćemo se na produkcijski server, spustiti aplikaciju, kopirat ćemo sve potrebne datoteke, podesiti konfiguraciju za produkciju i pokrenuti aplikaciju?
- Imamo osobu u timu koja ***zna*** kako se aplikacija pušta produkciju i to je uvijek njen zadatak?

CD - Continuous Delivery

- Continuous Delivery je način razvoja softvera kod kojeg se softver može pustiti u produkciju bilo kada.
  - To ne znači da, iako smo u mogućnosti, obavezno moramo softver puštati u produkciju vrlo često.
  - Continuous Deployment je način razvoja kod kojeg svaka promjena automatizmom ide u produkciju (ako prođe sve automatizirane provjere).
- Potrebno je automatizirati sve nakon što je kod *commit-an*. Put od *commit-a* do produkcije se može ponoviti bez intervencije ljudi.
- To je *deployment pipeline* – automatizirana implementacija *build, deploy, test & release* postupka za pojedini softver.

CD - Continuous Delivery

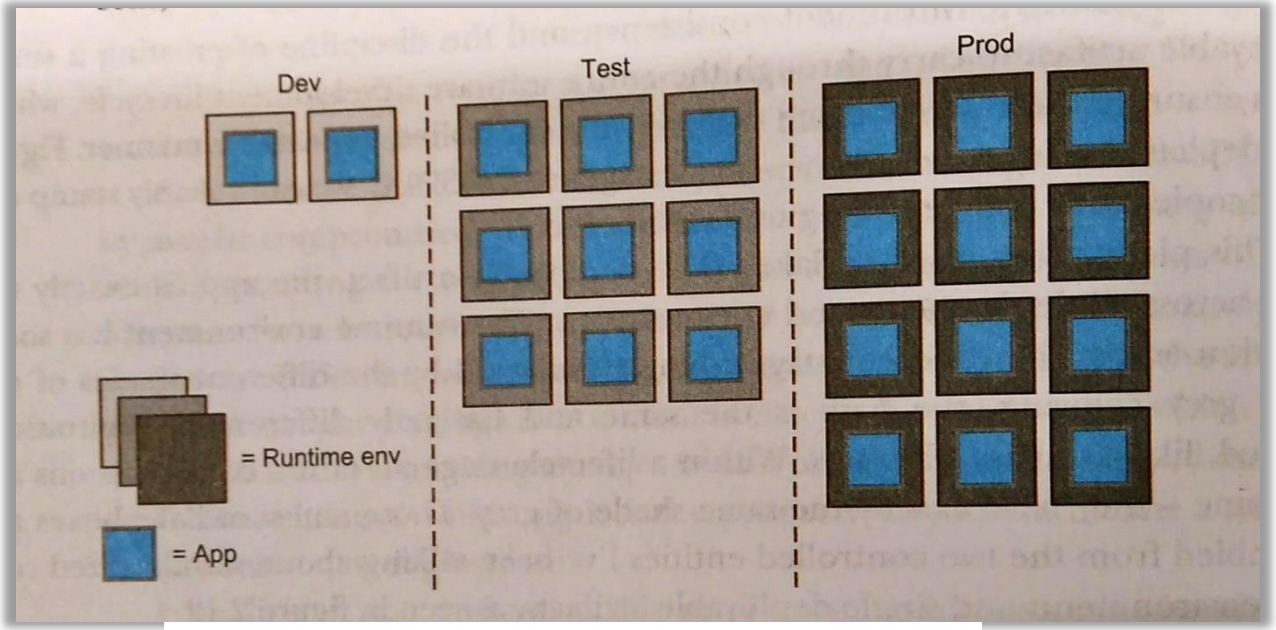
- Principi isporuke softvera:

1. Kreiraj ponovljiv i pouzdan postupak isporuke softvera.
2. Automatiziraj gotovo sve.
3. Sve čuvaj u *Version Control.*
4. Ono što je teško radi što češće kako bi što prije riješio problem.
5. *Build Quality In (functional test, stress test, acceptance test, …).*
6. Odrađeno znači da je funkcionalnost u produkciji. Odrađeno znači da je funkcionalnost ukinuta iz produkcije (jer nije više potrebna).
7. Svi su odgovorni za postupak isporuke softvera.

- Ključne prednosti:

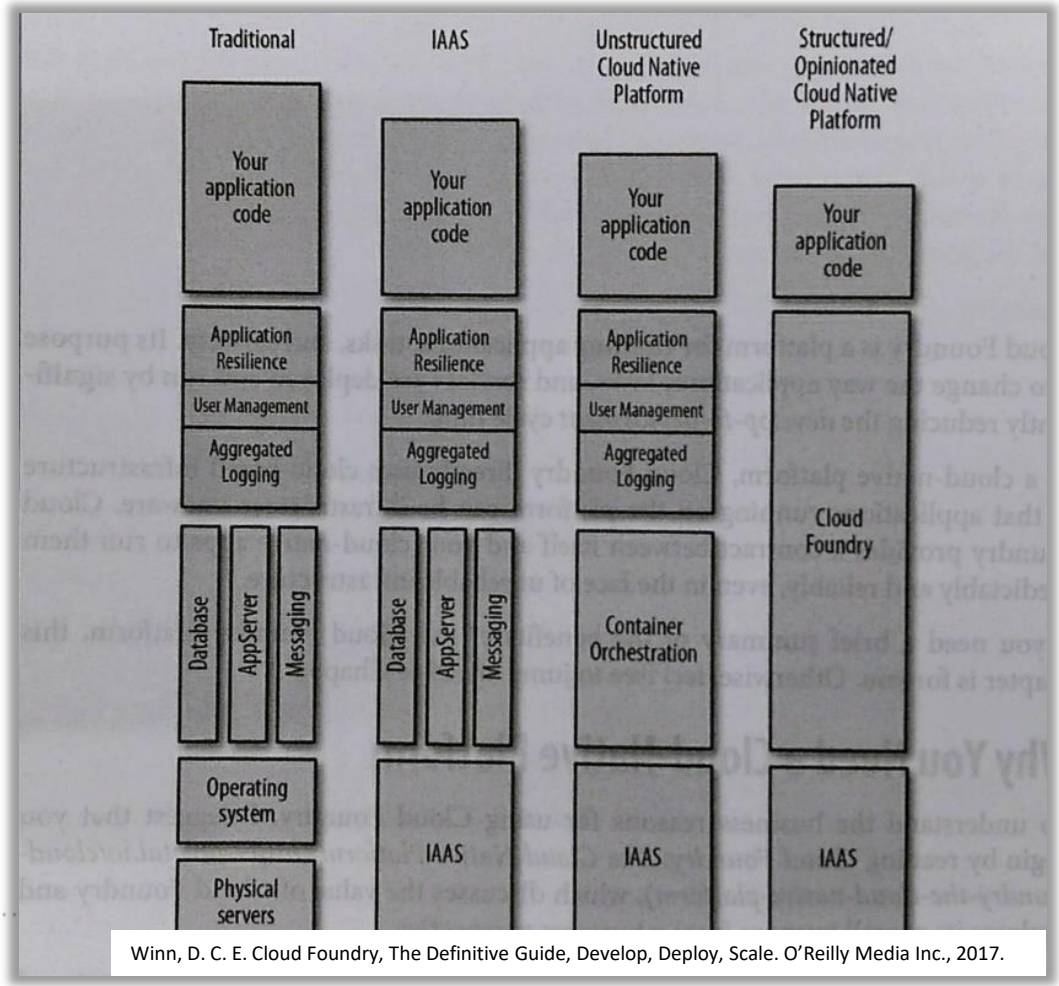
1. Smanjen rizik puštanja u produkciju.
2. Vidljiv i pouzdan napredak – ono što je u produkciji je očito funkcionalno.
3. Povratna informacija od korisnika – što prije korisnici počnu koristiti softver prije ćemo znati radimo li nešto što je njima korisno ili ne.

Kakvo nam je okruženje potrebno?



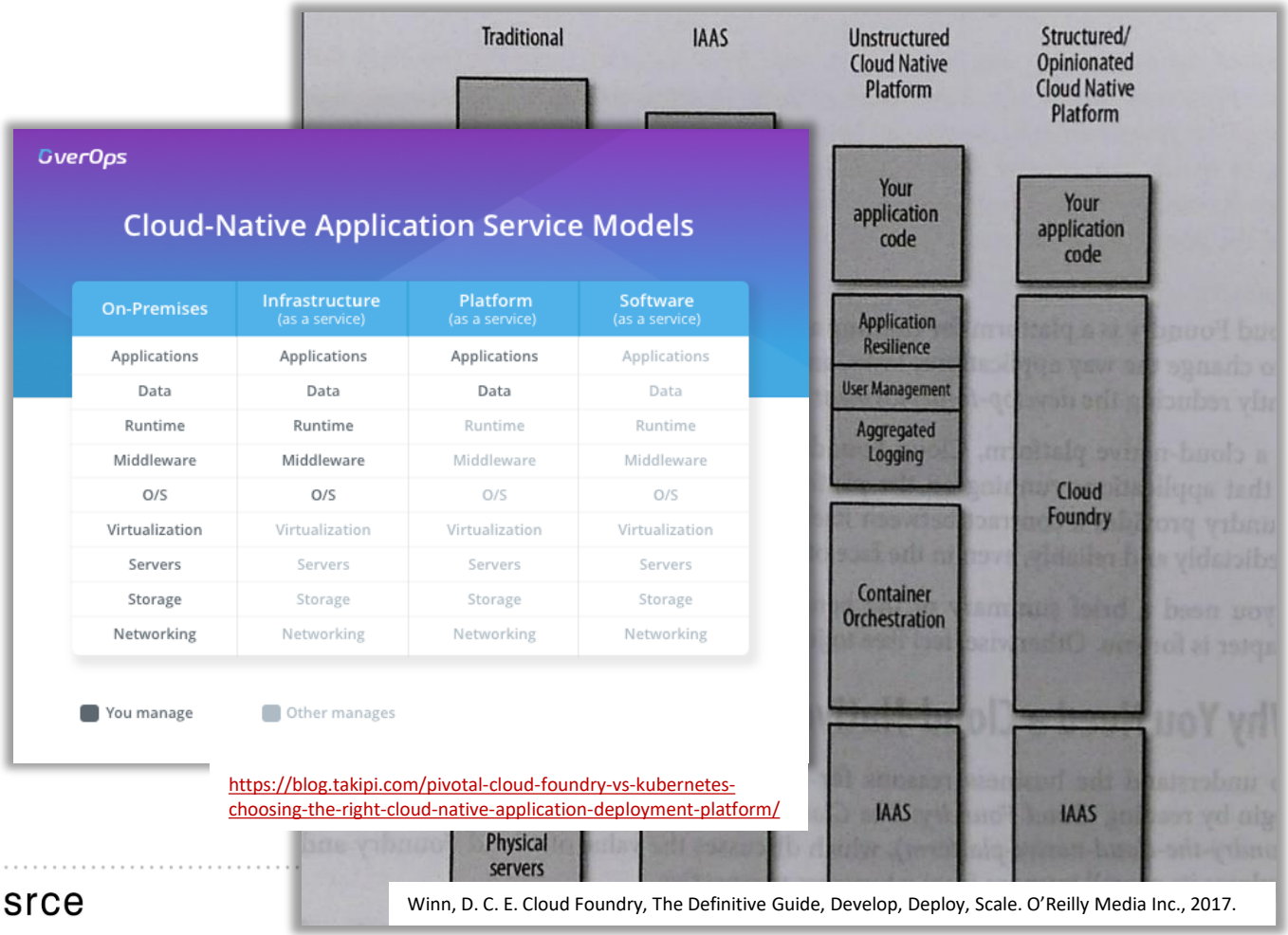
Davis, C. Cloud Native Foundations. Manning Publications Co., 2017.

Na kojoj platformi ćemo to realizirati?



Winn, D. C. E. Cloud Foundry, The Definitive Guide, Develop, Deploy, Scale. O'Reilly Media Inc., 2017.

Na kojoj platformi ćemo to realizirati?



OverOps

Cloud-Native Application Service Models

On-Premises	Infrastructure (as a service)	Platform (as a service)	Software (as a service)
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

You manage Other manages

https://blog.takipi.com/pivotal-cloud-foundry-vs-kubernetes-choosing-the-right-cloud-native-application-deployment-platform/

Winn, D. C. E. Cloud Foundry, The Definitive Guide, Develop, Deploy, Scale. O'Reilly Media Inc., 2017.

Cloud u Srcu?

- Što je Cloud Computing? SaaS, PaaS, IaaS?
- Cloud Computing načela:
  - Elasticity – The ability to handle concurrent growth through dynamically scaling the service up and down at speed.
  - On demand – The ability to choose when and how to consume the required service.
  - Self-service – The ability to directly provision or obtain the required service without time-consuming ticketing.
- SaaS (software as a service) – daje mogućnost korištenja nekog softvera po potrebi bez nabavljanja, licenciranja i instalacije.
- IaaS (infrastructure as a service) – daje mogućnost korištenja mrežnih, podatkovnih, procesnih i memorijskih resursa u oblaku.

Hoće li naše aplikacije raditi u cloud okruženju?

The twelve-factor app

- Metodologija za izradu web aplikacija tj. *cloud-native application architectures*

- *Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;*
- *Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;*
- *Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;*
- ***Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;*
- *And can **scale up** without significant changes to tooling, architecture, or development practices.*

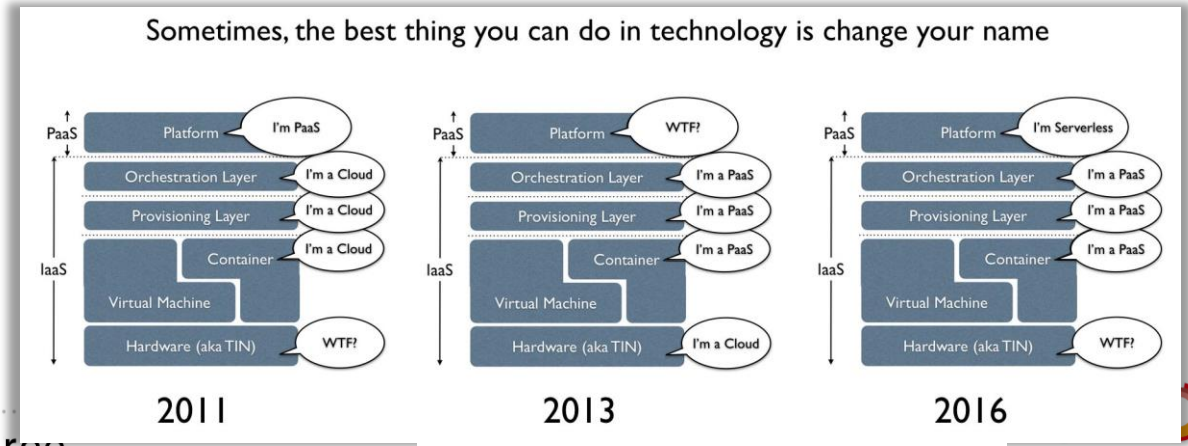
- Aplikacije moraju biti *stateless*.
  - Stanje aplikacije (npr. *http session*) mora se pohraniti izvan aplikacije (*in-memory data grids, caches, …*).
  - Takve aplikacije mogu se brzo kreirati i uništavati, mogu se spajati i odspajati od sustava u kojima je pohranjeno stanje aplikacije.

The twelve-factor app (https://12factor.net/)

- **I. Codebase**
- One codebase tracked in revision control, many deploys
- **II. Dependencies**
- Explicitly declare and isolate dependencies
- **III. Config**
- Store config in the environment
- **IV. Backing services**
- Treat backing services as attached resources
- **V. Build, release, run**
- Strictly separate build and run stages
- **VI. Processes**
- Execute the app as one or more stateless processes
- **VII. Port binding**
- Export services via port binding
- **VIII. Concurrency**
- Scale out via the process model
- **IX. Disposability**
- Maximize robustness with fast startup and graceful shutdown
- **X. Dev/prod parity**
- Keep development, staging, and production as similar as possible
- **XI. Logs**
- Treat logs as event streams
- **XII. Admin processes**
- Run admin/management tasks as one-off processes

PaaS u Srcu?

- Što je Platform as a Service (PaaS)?
  - *Middleware-provisioning systems?*
  - *Configuration-management?*
  - *Container-management and orchestration?*
- Gdje je granica između IaaS-a, PaaS-a i SaaS-a?
- Uobičajena ograničenja PaaS-a:
  - Nemogućnost nadzora platforme.
  - Nemogućnost nadogradnje platforme.
  - Ograničena podrška za programske jezike.
  - *Lock-in* i nemogućnost kreiranja privatnog PaaS-a.

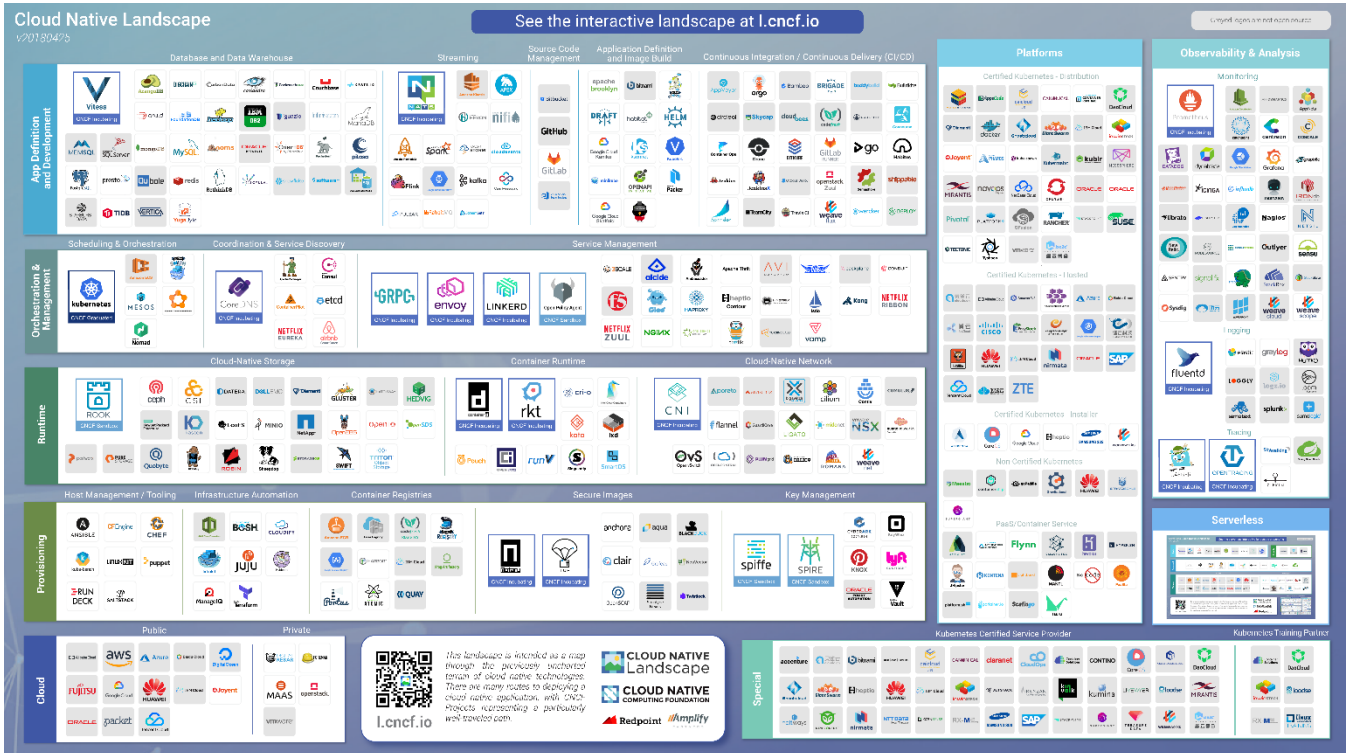


https://twitter.com/swardley/status/952180307540824065

Cloud-Native Platform u Srcu?

- Cloud-Native Platform nudi više funkcionalnosti od uobičajenog PaaS-a:
  - Elastičnost i otpornost na pogreške.
  - Sigurnost.
  - Podrška za životni ciklus aplikacije.
  - Nadzor rada aplikacije i platforme kroz agregirane *stream-ove* logova i metrika.
  - …
- „Cloud-Native Platforme su dizajnirane da rade više za programera kako bi se on mogao baviti onim što je važno.”
  - pouzdano i na predvidljiv način.
- To može omogućiti da se ciklus *build, test, deploy* kao i skaliranje obavlja bitno brže.

Pregled tržišta



https://github.com/cncf/landscape

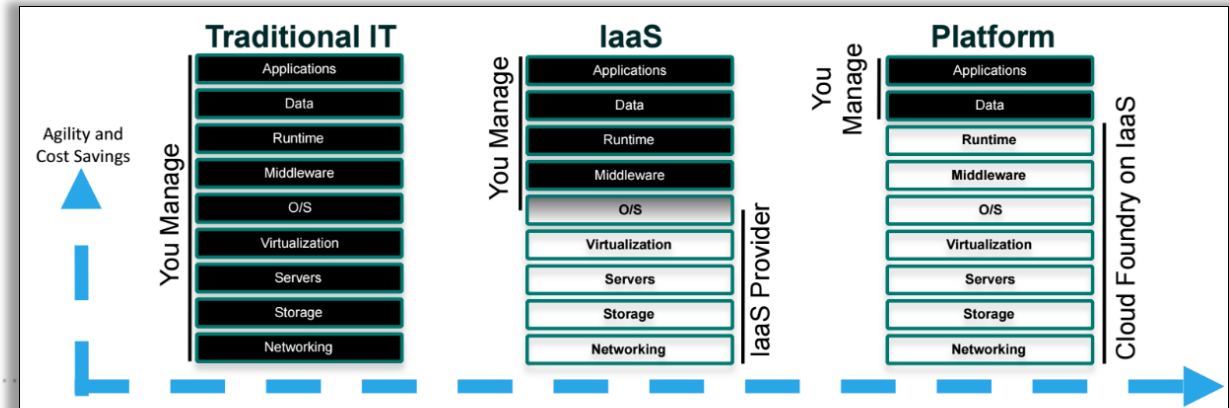
Cloud Foundry (CF) u Srcu?

- Cloud Foundry definira odnos između:
  - Infrastrukturnog sloja koji ga podupire
  - Aplikacija i servisa koje podržava
- Cloud Foundry ima tri ključne karakteristike:
  - Strukturirana – pruža (na ponovljiv način) sve što je potrebno za rad aplikacije bez obzira na infrastrukturu na kojoj radi. Suprotno od „*build your own platform*".
  - Samouvjerena *(„opinionated")* – radi određene pretpostavke i optimizacije kako bi se smanjila kompleksnost. Osigurava konzistentnost kroz sva okruženja bez potrebe za detaljnom i obaveznom konfiguracijom (iako je i to moguće).
  - Otvorena – može raditi na raznim IaaS-ovima, omogućava korištenje brojnih razvojnih *framework-a*, programskih jezika i servisa te je *open-source* (*Apache 2 licence*)
- *The Application as the Unit of Deployment (buildpacks)*
- *Supports Docker as a first-class citizen*

Cloud Foundry

- Open-source verzija
  - https://www.cloudfoundry.org/
  - https://github.com/cloudfoundry/cf-release
- 7 certificiranih distribucija
- 9 necertificiranih distribucija
- 8 podržanih IaaS-a

- Veliki broj podržanih servisa i integracija
- 9 službeno podržanih jezika *buildpack-a (*Supported Languages, Frameworks, and Technologies*)*
  - Veliki broj *community created (https://github.com/cloudfoundry-community/cf-docs-contrib/wiki/Buildpacks)*



https://docs.pivotal.io/pivotalcf/2-0/concepts/overview.html

Cloud Foundry vs. Kubernetes
"Application" PaaS vs. "Container" PaaS

- *These products offer a higher level of abstraction than we get from IaaS products meaning that, beyond networking, storage and servers, the application's O/S, middleware and runtime are all managed by the PaaS.*
- *They are both open source cloud PaaS products for building, deploying and scaling applications.*

- *CF is a high-level abstraction of cloud-native application development. You give CF an application, and the platform does the rest. It does everything from understanding application dependencies to container building and scaling and wiring up networking and routing.*
- *Cloud Foundry's platform is a higher-level abstraction and so it offers a higher level of productivity to its users. With productivity, though, comes certain limitations in what can be customized in the runtime.*
- *CF is ideal for new applications, cloud-native apps and apps that run fine out of a buildpack. For teams working with short lifecycles and frequent releases, CF offers an excellent product.*

- *Kubernetes is a container scheduler or orchestrator. With container orchestration tools, the user creates and maintains the container themselves.*
- *Instead of focusing only on the app, the developer needs to create the container and then maintain it in the future, for example, when anything on the stack has an update (a new JVM version, etc.).*
- *Kubernetes is a lower-level abstraction in the PaaS world meaning greater flexibility to implement customizations and build your containers to run how you want them to run.*
- *When moving to any new system or product, a **good rule of thumb is to use the highest level abstraction that will solve your problem** without putting any unnecessary limitations on the workload.*

https://blog.takipi.com/pivotal-cloud-foundry-vs-kubernetes-choosing-the-right-cloud-native-application-deployment-platform/

https://medium.com/@odedia/comparing-kubernetes-to-pivotal-cloud-foundry-a-developers-perspective-6d40a911f257

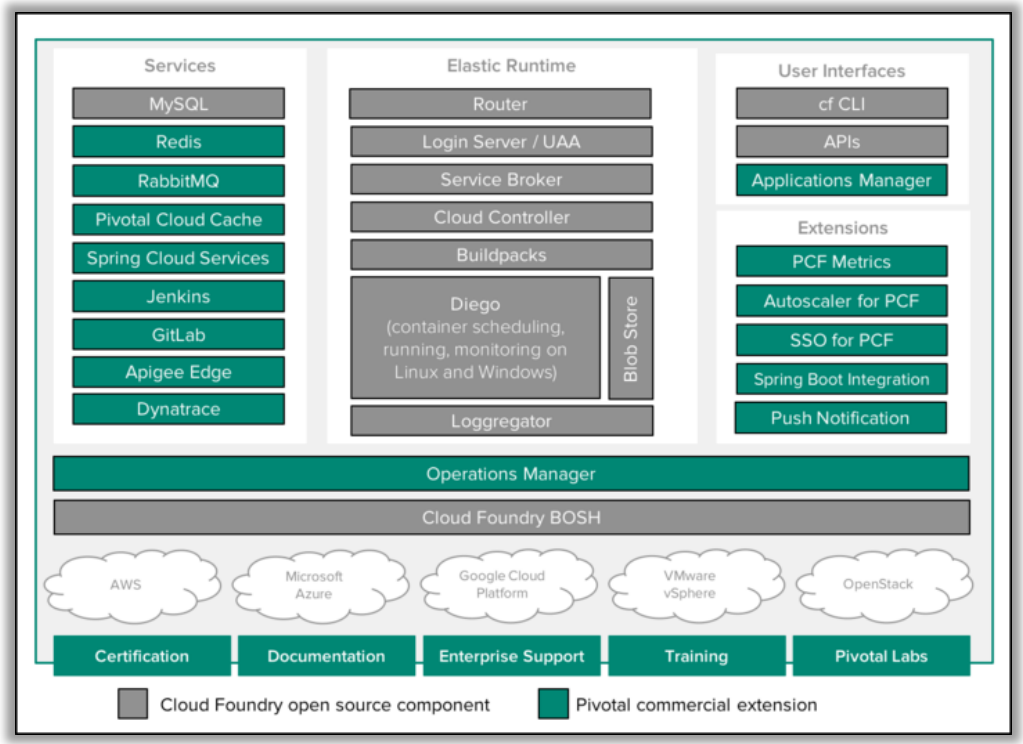
Pivotal Cloud Foundry (PCF) 2.0 u Srcu?

- Komercijalna certificirana distribucija (*https://pivotal.io/platform*)

- Pivotal Application Service (PAS) - *platform for deploying applications that have been built using cloud native/12 factor app patterns*
- Pivotal Container Service (PKS) - *provides infrastructure management for the open source version of Kubernetes*
- Pivotal Function Service (PFS) - *serverless platform (preview/beta)*

- Dio toga što nudi dodatnog u odnosu na CF:
  - Jednostavnije upravljanje infrastrukturom (Ops Manager)
  - Proširena funkcionalnost (Apps Manager, PCF Metrics, …)
  - Dodatni servisi za programere i aplikacije (npr. *highly available data and middleware services*)
  - Spring Cloud Services (*Cloud-native Enablement – Java/Spring & .NET*)
  - Enterprise Support
  - PCF Dev - *PCF Dev is a small footprint distribution of Pivotal Cloud Foundry (PCF) intended to be run locally on a developer machine…*

Pivotal Cloud Foundry (PCF) 2.0 u Srcu?



https://docs.pivotal.io/pivotalcf/2-0/concepts/overview.html

Zašto nam treba PaaS u Srcu?

Denis Kranjčec, Srce