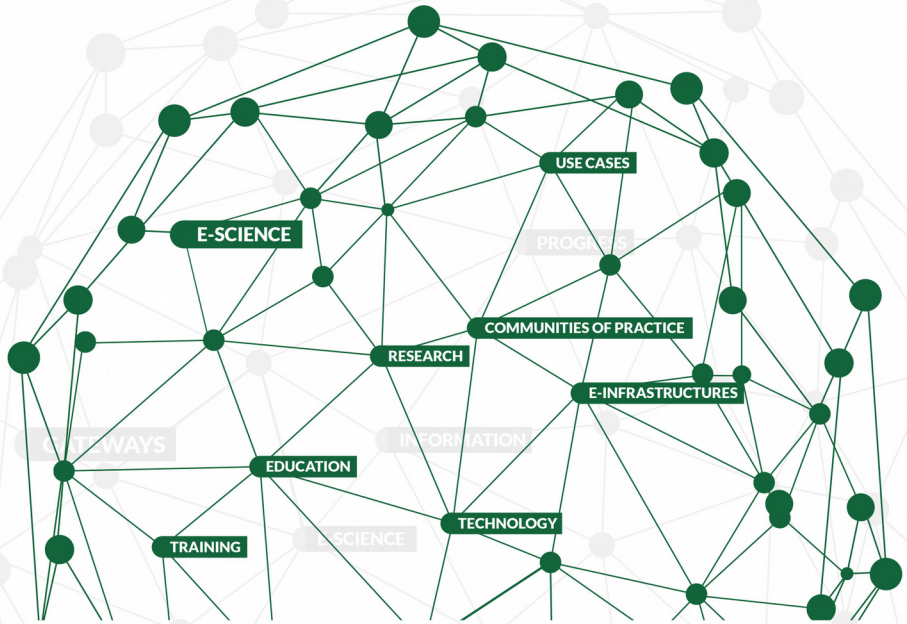
Patterns in e-Science Applications



Sci-GaIA
www.sci-gaia.eu

Bruce Becker, CSIR (South Africa) bbecker@csir.co.za
EthERNet e-Research Hackfest – Addis Ababa (Ethiopia)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n° 654237

Or...

What are you even building ?

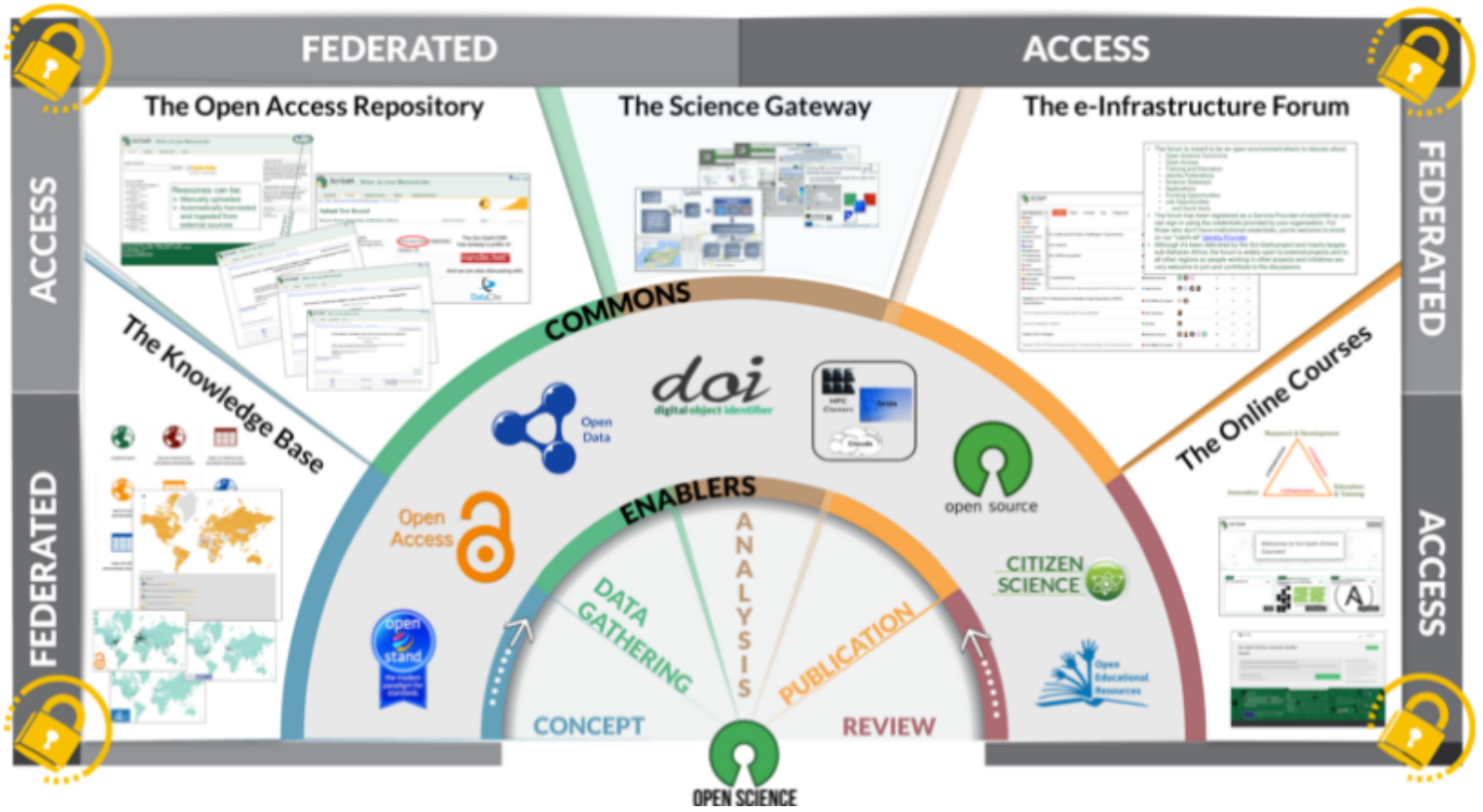
Sci-GaIA
www.sci-gaia.eu

Introduction

- Africa, like the web, is a big, wild place !
  - There are many many freedoms, but sometimes we need to work together to make things
- This is a very rapidly evolving environment – new applications, infrastructures and platforms are being delivered almost every day.
- A good situational awareness is necessary to make design and implementation choices which will serve you for the future.
- Just as you are hacking your applications, so we are trying to hack the hackfest – make it better and more streamlined every time.
- We want to not only talk about *existing* patterns, but also help identify new ones, as we see them arise

Sci-GaIA
www.sci-gaia.eu

A platform for Open Science



Sci-GaIA
www.sci-gaia.eu

Outline

- Why do we talk about patterns ?
- Development patterns for modern web apps :
  - The 12-factor application
  - Continuous Integration
- Patterns in e-Infrastructure
- Patterns in Research
- Integration Patterns

Why do we talk about patterns ?

“ In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. **A design pattern isn't a finished design that can be transformed directly into code.** It is a description or template for how to solve a problem that can be used in many different situations. ” - sourcemaking.com/design_patterns

Design can also follow “ antipatterns ” sourcemaking.com/antipatterns

We can use these as analogies for developing our applications

Why do we talk about patterns ?

- There is no “ right ” way to work on these projects
  - You have freedom in choosing the application language stack, implementation strategy, execution flow, etc
- However, by identifying the pattern which most closely describes the application you are working on, you can be better guided

In the “old days”

- High-performance or high-throughput research has typically been done via the command-line.
- Research workflows could be scripted and automated to a large extent, and good practices.
- Middleware stacks were written for the command-line to make it easier to perform distributed computing – typically grid computing
- The language of research was typically scripts –
  - shell, perl
  - Application stacks – ROOT, python, MATLAB/Octave
- The language of the web was originally static HTML, and the browser was essentially a client applications for reading data and information.
- Not that great for scientific work

Now

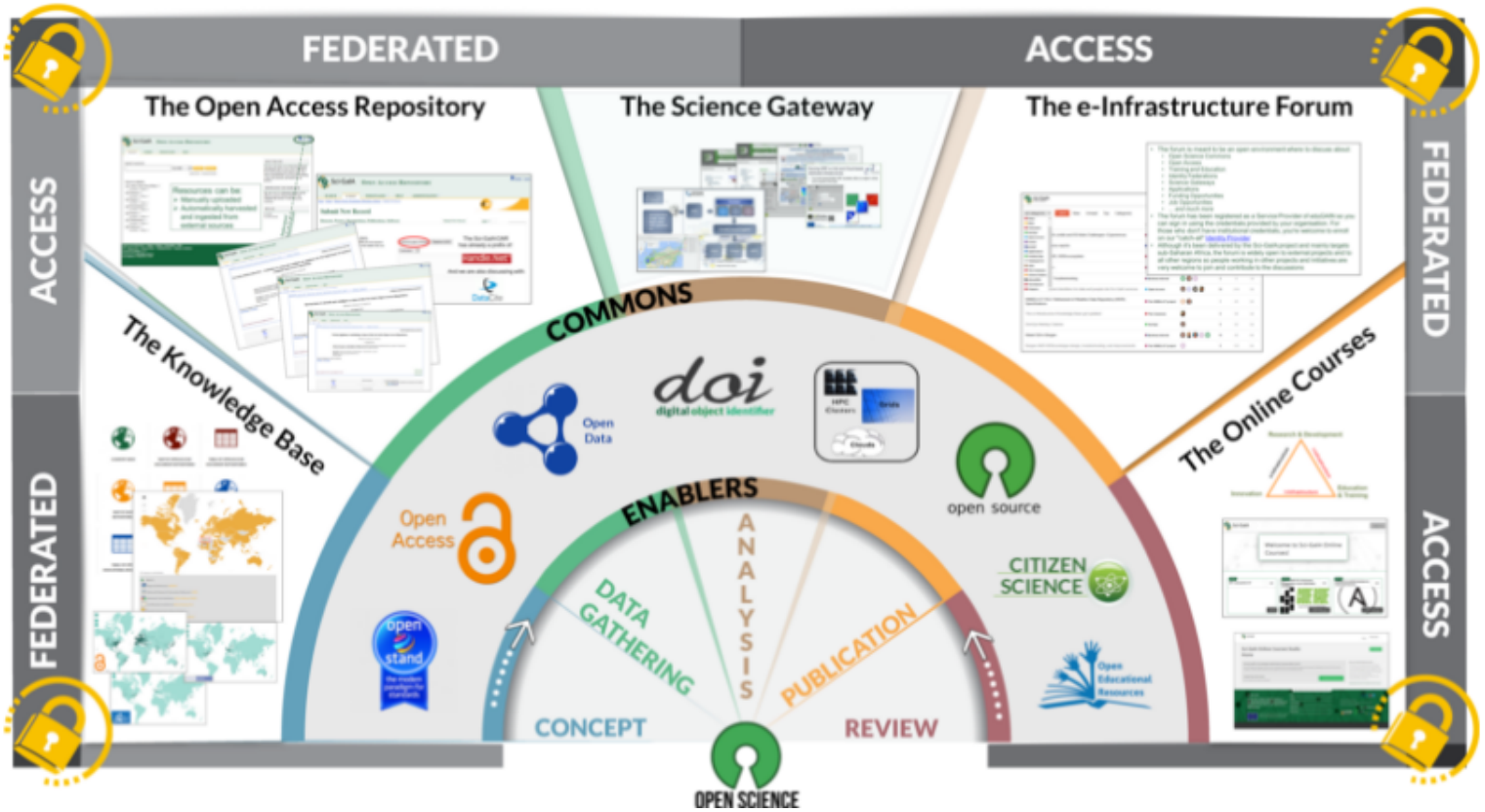
- Now we can run very rich applications right in the the browser
- The web is not just for reading, but for creating and collaborating
- Most features of the command-line can be replicated on the web
- The web has become one of the most important tools for scientific discovery and dissemination
- The web provides much of our public computing infrastructure
- However, we still need :
  - dedicated platforms for computing and data
  - specific application stack for scientific work

Sci-GaIA
www.sci-gaia.eu

Let’s build a web app !

The web as part of an Open Science Platform

www.sci-gaia.eu/osp



Sci-GaIA
www.sci-gaia.eu

But what *is* a web app ?

- The web is a pretty wild place - there is a lot of freedom to create just about anything !
- The *infrastructure* which a web application runs on has experienced rapid evolution
- However, over the last couple of years, some common patterns have emerged which describe good practice in developing web applications
- In particular the paradigm of cloud computing

Application patterns

12 factor applications

12factor.net

- 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

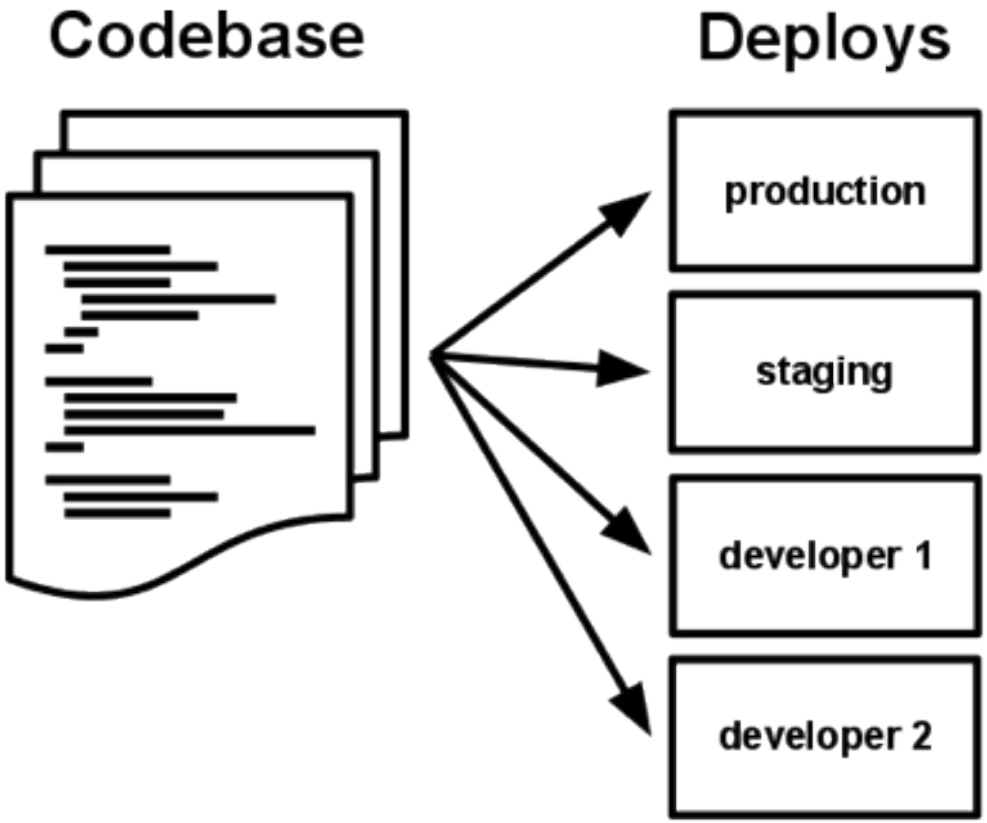
Sci-GaIA
www.sci-gaia.eu

1 : Code Base

12factor.net/codebase

- One codebase tracked in revision control, many deploys
- There is always a one-to-one correlation between the codebase and the app:
  - If there are multiple codebases, it's not an app – it's a distributed system. Each component in a distributed system is an app, and each can individually comply with twelve-factor.
  - Multiple apps sharing the same code is a violation of twelve-factor. The solution here is to factor shared code into libraries which can be included through the dependency manager.

12 Factor Applications – Code Base



2 : Dependencies

12factor.net/dependencies

- A twelve-factor app never relies on implicit existence of system-wide packages.
- It declares all dependencies, completely and exactly, via a dependency declaration manifest.
- Furthermore, it uses a dependency isolation tool during execution to ensure that no implicit dependencies “leak in” from the surrounding system.
- The full and explicit dependency specification is applied uniformly to both production and development.

3 : Config

12factor.net/config

- Store config in the environment
- An app's config is everything that is likely to **vary between deploys**
  - Resource handles to the database, other backing services
  - Credentials to external services
  - Per-deploy values such as the canonical hostname for the deploy
- Apps sometimes store config as **constants in the code**. This is a violation of twelve-factor, which requires strict **separation of config from code :** Config varies substantially across deploys, code does not.
- The twelve-factor app stores config in environment variables

4 : Backing services

12factor.net/backing-services

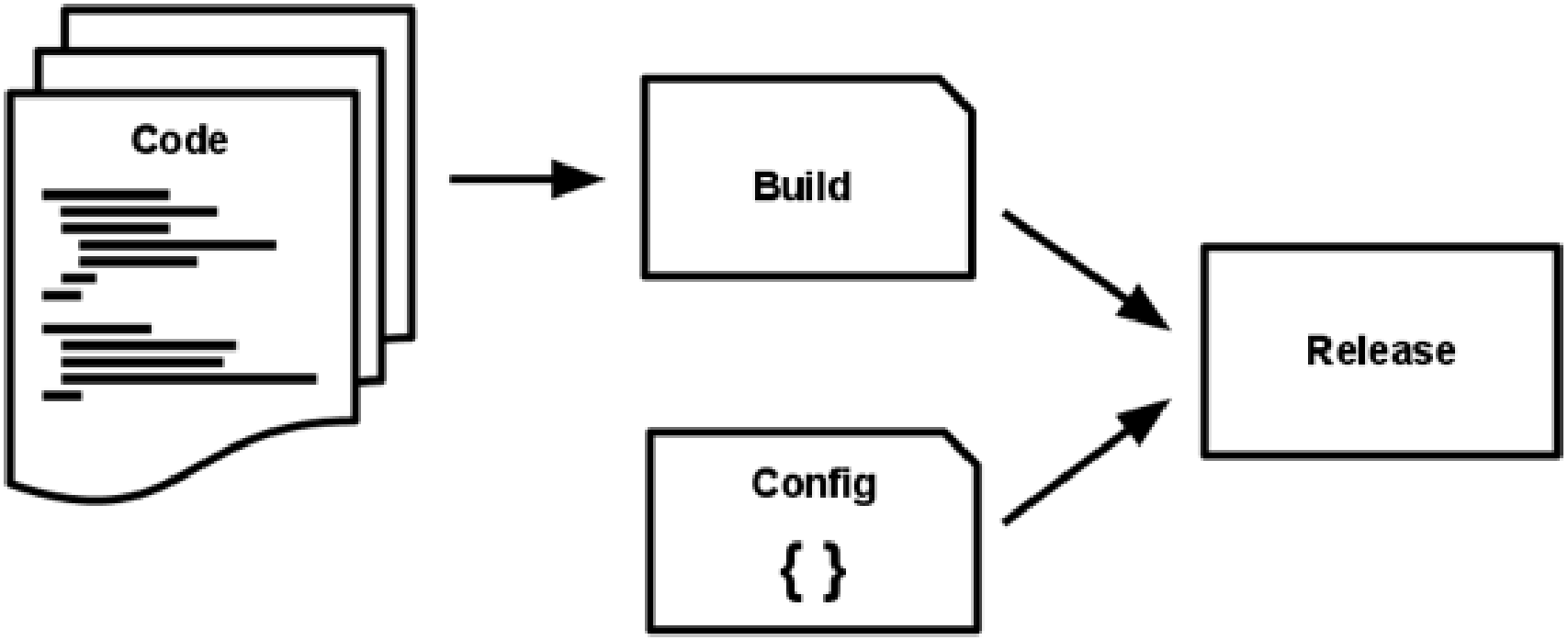
- Treat backing services as attached resources
- A backing service is any service the app consumes over the network as part of its normal operation. Examples include :
  - datastores (MySQL, CouchDB)
  - messaging/queueing systems
  - SMTP services for outbound email
  - caching systems
- In addition to these locally-managed services, the app may also have services provided and managed by third parties.
- The code for a twelve-factor app makes no distinction between local and third party services. To the app, both are attached resources, accessed via a URL or other locator/credentials stored in the config.

5 : Build, release, run

12factor.net/build-release-run

- The twelve-factor app uses strict separation between the build, release, and run stages
- A codebase is transformed into a (non-development) deploy through three stages:
  - The build stage is a transform which converts a code repo into an executable bundle known as a build. Using a version of the code at a commit specified by the deployment process, the build stage fetches vendors dependencies and compiles binaries and assets.
  - The release stage takes the build produced by the build stage and combines it with the deploy's current config. The resulting release contains both the build and the config and is ready for immediate execution in the execution environment.
  - The run stage (also known as "runtime") runs the app in the execution environment, by launching some set of the app's processes against a selected release.

12 Factor Applications : Build, Release, Run



10 : Dev/Prod parity

12factor.net/dev-prod-parity

- Keep development, staging, and production as similar as possible
- The twelve-factor app is designed for continuous deployment by keeping the gap between development and production small.
  - **The time gap**: A developer may work on code that takes days, weeks, or even months to go into production.
  - **The personnel gap**: Developers write code, ops engineers deploy it.
  - **The tools gap:** Developers may be using a stack like Nginx, SQLite, and OS X, while the production deploy uses Apache, MySQL, and Linux.

Continuous Integration

www.thoughtworks.com/continuous-integration

“Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

By integrating regularly, you can detect errors quickly, and locate them more easily.”

CI practices

- Maintain a single source repository
- Automate the build
- Make your build self-testing
- Every commit should build on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for anyone to get the latest executable
- Everyone can see what's happening
- Automate deployment

Execution and workflow models

Grid computing patterns

DOI: 10.1109/WORKS.2006.5282349
https://www.researchgate.net/publication/224602030_Parallel_computing_patterns_for_Grid_workflows

- “ Whereas a consensus has been reached on defining the set of workflow patterns for business process modeling languages, no such patterns exists for workflows applied to scientific computing on the Grid ”
- Usage patterns have emerged rather through re-use of community tools and due to restrictions placed on communities by the middleware stacks.
- For an older, fuller list of workflow patterns and tools, see Yu, J. & Buyya, R. J Grid Computing (2005) 3: 171
  DOI: 10.1007/s10723-005-9010-8

Grid workflow patterns

- Two basic kinds of execution models, with a few patterns – each with a few variants
- *Parallel* execution and *pipelined* execution
- There is significant difference between compute-parallelism and data parallelism

Grid parallelism

- Parallel execution patterns can be split into parallel execution and data parallelism patterns
- Compute-bound workflows are “ simple ” and can be partitioned depending on the nature of the dependencies between the tasks.
- Data-parallel patterns can be identified as :
  - Static data parallelism
  - Dynamic data parallelism
  - Adaptive data parallelism

Grid pipelined execution

- As opposed to parallel computing patterns, there are pipelined computing patterns which can be identified on grid infrastructures.
  - Best effort
  - Blocking
  - Buffered
  - SuperScalar
  - Streaming
- There are several tools which can be used to construct pipelines for execution on distributed computing platforms
  - See pditommaso/awesome-pipeline

Cloud usage patterns

www.cloudcomputingpatterns.org

- There are many, many ways to “ cloud ” science – but we can again identify recurring patterns :
- Cloud Workloads :
  - Static Workload
  - Periodic Workload
  - Once-in-a-lifetime Workload
  - Unpredictable Workload
  - Continuously Changing Workload
- Where grids and HPC’s work very well under certain static, periodic and once-off workloads, clouds typically designed to deal with widely-varying workloads

Research patterns

Sci-GaIA
www.sci-gaia.eu

Open Science

www.fosteropenscience.eu | www.openingscience.org

- Open Science is a means, not an end
- There are several *aspects* of Open Science which we need to consider when developing research applications
  - Reproducibility
  - Open Peer Review
  - Open Access
  - *etc*
- In order to be able to reproduce and validate research, acess to the e-Infrastructures needed to perform that research is necessary (cloud/grid/HPC)
- Data is key

Sci-GaIA
www.sci-gaia.eu

FAIR data

Findable, Accessible, Interoperable, and Re-usable.
www.force11.org/group/fairgroup/fairprinciples

- The Open Science paradigm relies crucially on data
- The Open Data (similar to the Open Source) definition is not entirely applicable
- FAIR data *can* be Open Data and vice-versa. Both imply
  - Accessibility
  - Interoperability
  - Re-usability
- However, FAIR data is more applicable to research data, whilst Open Data is more applicable to public data.

Sci-GaIA
www.sci-gaia.eu

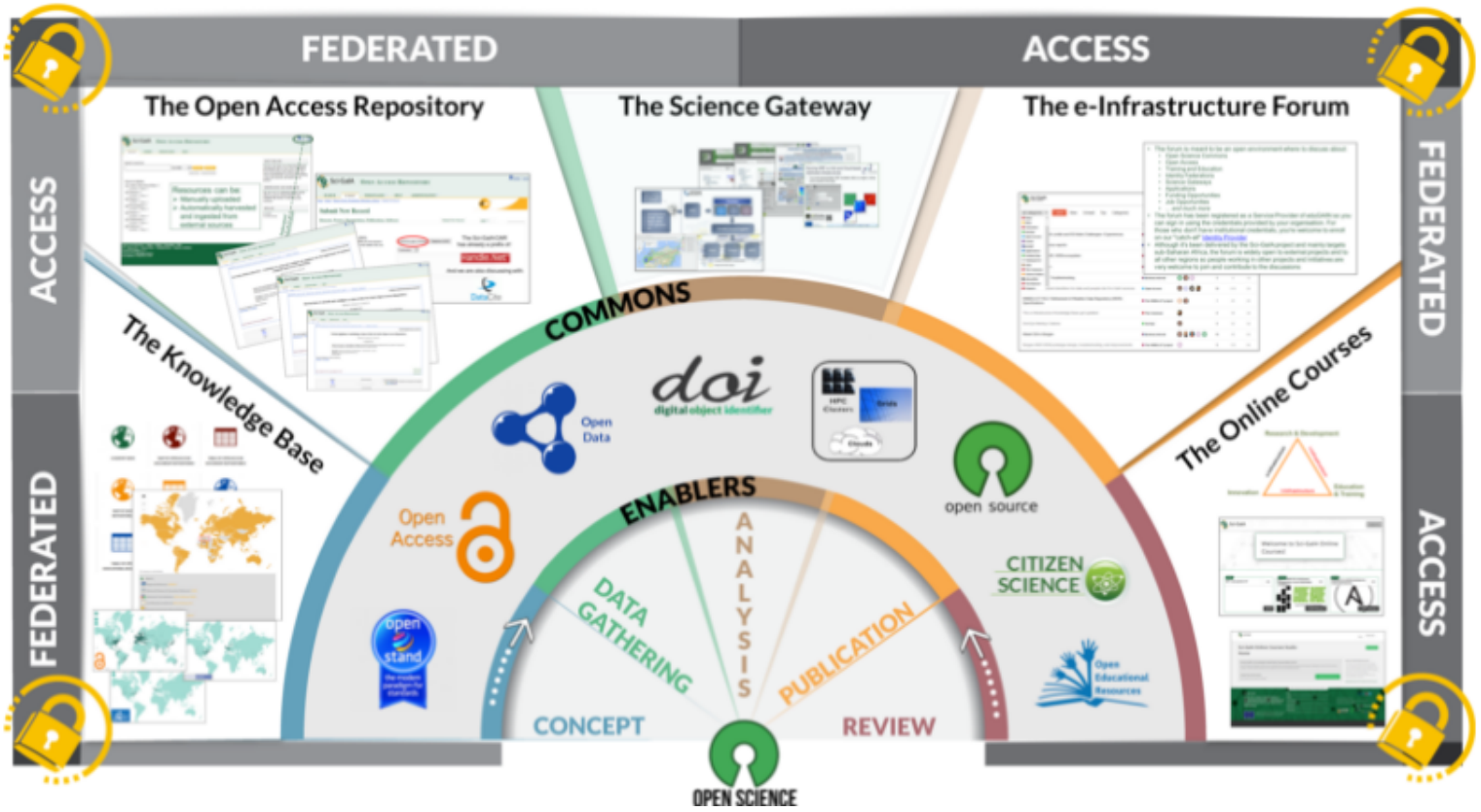
FAIR data

- To be findable :
  - F1. (meta)data are assigned a globally unique and eternally persistent identifier.
  - F2. data are described with rich metadata.
  - F3. (meta)data are registered or indexed in a searchable resource.
  - F4. metadata specify the data identifier.
- To be Accessible:
  - A1 (meta)data are retrievable by their identifier using a standardized communications protocol.
  - A1.1 the protocol is open, free, and universally implementable.
  - A1.2 the protocol allows for an authentication and authorization procedure, where necessary.
  - A2 metadata are accessible, even when the data are no longer available.

FAIR data

- To be Interoperable:
  - I1. (meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
  - I2. (meta)data use vocabularies that follow FAIR principles.
  - I3. (meta)data include qualified references to other (meta)data.
- To be Re-usable:
  - R1. meta(data) have a plurality of accurate and relevant attributes.
  - R1.1. (meta)data are released with a clear and accessible data usage license.
  - R1.2. (meta)data are associated with their provenance.
  - R1.3. (meta)data meet domain-relevant community standards.

A closer look at the platform



Sci-GaIA
www.sci-gaia.eu

Infrastructure Services

- Users
  - Authentication
  - Authorisation
  - ID management
- Compute
  - Workload management
  - Compute management
- Data
  - Movement
  - Storage
  - Management
- Others : Accounting, monitoring, publication, security, etc

Sci-GaIA
www.sci-gaia.eu

So, where can I get all these services ?

- No single infrastructure will do everything you want
- No single infrastructures can handle all usage patterns.
- However,
  - by adopting Open Standards and
  - building 12-factor applications

  You can more easily develop Open Science applications
- Don’t try to do everything on your own.

Sci-GaIA
www.sci-gaia.eu

EGI Service Catalogue

www.egi.eu/services

- Compute
  - High-Throughput Computing (grid computing)
  - Cloud Compute (federated cloud)
  - Cloud Container Compute (containers on FedCloud)
- Storage and Data
  - Online storage (grid storage)
  - Offline storage (archival)
  - Data movement

Sci-GaIA
www.sci-gaia.eu

Grid Infrastructure Services

www.africa-grid.org

- AfricaGrid : a collaboration to provide a platform for collaboration to researchers across Africa
- A peer of the EGI platform
- Provides compute and data services for individuals and research communities
- Cloud infrastructure :
  - AfricaGrid does not provide a *unified* cloud infrastructure, but several institutes are working on joining FedCloud.
- These are *open infrastructure*

Sci-GaIA
www.sci-gaia.eu

Guidelines and considerations

- The most important aspect to remember
  - 12 factor applications
  - Open Standards
- Open Standards
  - Consume services via REST APIs as far as possible
  - We will be using the SAGA standard to connect the Science Gateway to several different compute infrastructures
  - Just because we support standards-based access to infrastructure, doesn't mean that infrastructure is magically available
  - Try to estimate usage patterns

Integration patterns

- Everything = code
  - Try to respect the 12 factor application as far as possible
  - Reduce reliance on “magic wands” : reduce manual intervention.
- Automate Everything
  - The more places your app can be deployed and run, the better your chances of people using it.
  - Make deployment and configuration automatic, especially regarding backing services
- Robust by design :
  - Reduce the number of “ moving parts ”
  - Identify and reduce single points of failure
  - TEST. EVERYTHING.

Integration Patterns

- Use the infrastructure as backing services
  - For the “ heavy-lifting ” compute/data services, use the standard adaptors
  - Consider the data parallelism of your research
  - Move data only when necessary
  - Use Identity Federations for authentication, and dedicated services for authorisation.
- Focus on workflow
  - Try to identify which kinds of grid or cloud compute patterns will be used
  - Write the web app to guide the user in these workflows
  - Don’t make the user do unnecessary tasks

Discussion