



Projektplan

Silvan Adrian
Fabian Binna

1 Änderungshistorie

Datum	Version	Änderung	Autor
17.09.15	1.00	Erstellung des Dokuments	Gruppe
18.09.15	1.01	Organisationsstruktur + Einführung + Projektübersicht	Silvan Adrian
18.09.15	1.02	Qualitätsmassnahmen	Fabian Binna
20.09.15	1.03	Text verbessert und zusätzliche Infos eingefügt	Silvan Adrian
25.09.15	1.04	Verbesserungen	Silvan Adrian
25.09.15	1.05	Sprint Planungsübersicht eingefügt	Silvan Adrian
26.09.15	1.06	Testing erweitert/verbessert + Logging	Fabian Binna
02.10.15	1.07	Verbesserungen gemäss neusten Erkenntnissen	Silvan Adrian

Inhaltsverzeichnis

1	Änderungshistorie	2
2	Einführung	5
2.1	Zweck	5
2.2	Gültigkeitsbereich	5
2.3	Referenzen	5
2.3.1	APIs:	5
2.3.2	SCRUM Guide:	5
3	Projektübersicht	5
3.1	Zweck und Ziel	5
3.2	Primäre Features	5
3.3	Erweiterte Features	6
3.4	Lieferumfang	6
3.5	Annahmen und Einschränkungen	6
4	Projektorganisation	6
4.1	Organisationsstruktur	7
4.2	Externe Schnittstellen	7
5	Managment Abläufe	7
5.1	Kostenvoranschlag	7
5.2	Sprints	8
5.3	Besprechungen	8
6	Risikomanagement	8
6.1	Risiken	8
6.2	Umgang mit Risiken	8
7	Arbeitspakete	8
8	Infrastruktur	9
8.1	Entwicklungsinfrastruktur	9
8.2	Tools/Software	9
8.3	Kommunikationsmittel	9
9	Qualitätsmassnahmen	9
9.1	Dokumentation	9
9.2	Projektmanagement	9
9.3	Entwicklung	10
9.3.1	Unit Testing / Test-Driven Development	10
9.3.2	Code Review	10

9.3.3	Logging	10
9.3.4	Metrikanalyse	10
9.3.5	Code Style Guidelines	10
9.4	Testen	10
9.5	Modultest	10
9.6	Integrationstest	11
9.7	Systemtest	11
9.8	Abnahmetest	11

2 Einführung

2.1 Zweck

Dieses Dokument beschreibt die Planung der SA SDDC.

2.2 Gültigkeitsbereich

Dieses Dokument ist während des ganzen Projekts gültig und wird laufend aktualisiert.

2.3 Referenzen

2.3.1 APIs:

<http://developer.openstack.org/>

2.3.2 SCRUM Guide:

<http://www.scrumguides.org>

3 Projektübersicht

Es soll eine generische API erstellt oder eine bestehende API verwendet werden, um über ein Dashboard direkt Services zu abonnieren, dabei können bei der API Vorlagen hinterlegt werden, welche mehrere Servicemodule beinhalten (Compute,Storage,Network). Die API soll dabei möglichst modular aufgebaut sein, damit Neuerungen einfach auf den neusten Stand gebracht werden können.

3.1 Zweck und Ziel

Gelerntes aus verschiedenen Modulen anwenden, sich mit der gestellten Aufgabenstellung auseinandersetzen und diese umsetzen.

3.2 Primäre Features

- Generische API
- Unterstützung von Private Cloud Anbietern
- Unterstützung von Docker
- Unterstützung von ausgewählten Public Cloud Anbietern

3.3 Erweiterte Features

- Spartanisches User-Dashboard
- Unterstützung für mehr Public Cloud Anbieter
- Admin-Dashboard um Funktionen erweitern und verbessern

3.4 Lieferumfang

- Source-Code (Dashboard/API)
- Dokumentation (Projekt und Software)

3.5 Annahmen und Einschränkungen

Die Applikation setzte sich aus einem Admin-Dashboard und einer generischen API zusammen, beides sollte dabei verteilt betrieben werden können. Das User-Dashboard würde lediglich zum Testen der API verwendet. Die generische API soll dabei so viele Anbieter wie möglich abdecken.

4 Projektorganisation

Wir setzen beim Project auf Scrum, wodurch lediglich die Projektrollen Product Owner, Entwickler und Scrum-Master vorhanden sind.

4.1 Organisationsstruktur



4.2 Externe Schnittstellen

Projektbetreuer:

Beat Stettler (Beat.Stettler@ins.hsr.ch)

Urs Baumann (Urs.Baumann@ins.hsr.ch)

5 Managment Abläufe

5.1 Kostenvoranschlag

Gemäss Vorgaben wird mit 480 Stunden gesamt gerechnet + die einberechneten Risiken von 41 Stunden.

Gesamtaufwand: 521 Stunden

Risikoaufwand: 41 Stunden

Aufwand pro Woche: ca.: 37 Stunden

Aufwand pro Woche pro Teammitglied: ca.: 19 Stunden

5.2 Sprints

Sprint	Beschreibung	Beginn	Ende
Sprint 1	Grundgerüst API + Workflow, API ansprechbar und vorgefertigte Services können abonniert werden	12.10.15	26.10.15
Sprint 2	Restful Interface für die API, um Services über HTTP abonnieren zu können	26.10.15	09.11.15
Sprint 3	Authentifizierung bei API für Client Applikation + Benutzer Authentifizierung für jeweiligen Cloud Anbieter	09.11.15	23.11.15
Sprint 4	Admin-Dashboard + User-Dashboard erstellen (sehr spartanisch) und letzter Release der Software	23.11.15	7.12.15

5.3 Besprechungen

Daily Meetings (ca. 10 - 15min) werden während der Pause oder über Skype stattfinden, um sich gegenseitig über Probleme, abgeschlossene Arbeitspakete etc. zu informieren. Nach jedem Abschluss eines Sprints wird ein Sprintreview geplant und durchgeführt, um zu prüfen ob alles gemäss Plan erledigt wurde + die aktuellste Version der Software released.

6 Risikomanagement

6.1 Risiken

Risiken werden im Dokument Initiales_Risikomanagement.pdf beschrieben

6.2 Umgang mit Risiken

Für die Risiken werden Reserven eingeplant. Die Reserven werden direkt in die einzelnen Tickets eingerechnet. Falls Risiken eintreffen werden diese sofort an einem der Daily-Meetings kommuniziert und mögliche Lösungen evaluiert.

7 Arbeitspakete

Die Arbeitspakete werden in Open Project erstellt und gepflegt. Lesender Zugriff ist anonym möglich, schreibender nur eingeloggt (Projekt ist öffentlich).

Link zur Open Project Instanz : <http://sddc.silvn.com>

8 Infrastruktur

8.1 Entwicklungsinfrastruktur

Name	Hardware	Betriebssystem	IDE
Silvan Adrian	MacBook Pro	OSX 10.10.5	pyCharm CE 4.5
Fabian Binna	Lenovo T430s	Windows 10	pyCharm CE 4.5

8.2 Tools/Software

- **BuildServer:** Travis-CI
- **Versionsmanagement:** GIT
- **Notifications:** Slack
- **Wireframing:** Pencil
- **Code Coverage:** Coverage.py

8.3 Kommunikationsmittel

- E-Mail
- Skype
- Open Project (Kommentare)
- GitHub (Issues, Kommentare)
- Whatsapp
- Slack

9 Qualitätsmassnahmen

9.1 Dokumentation

Die Dokumentation befindet sich auf einem privaten GitHub Repository. Die Texte werden in LaTeX geschrieben. Die Dokumente werden versioniert.

https://github.com/silvanadrian/SDDC_Doku.git

9.2 Projektmanagement

Für das Projektmanagement wird OpenProject verwendet.

sddc.silvn.com

9.3 Entwicklung

9.3.1 Unit Testing / Test-Driven Development

Die Unit Tests kommen in einen separaten Ordner "Test". Es wird eine möglichst hohe Code Coverage angestrebt. Die Code Coverage wird mit einem Tool sichergestellt.

Die Klassen werden mit Hilfe von Test-Driven Development implementiert.

9.3.2 Code Review

Nach jedem Sprint oder bei Abschluss grosser Arbeitspakete wird ein Code Review durchgeführt.

Die Review Protokolle werden auf dem GitHub Repository SDDC-Doku abgelegt.

9.3.3 Logging

Ein Logging-System unterstützt die Entwicklung und später das Operating.

TRACE: Ausführliches Debugging

DEBUG: allgemeines Debugging

INFO: allgemeine Informationen

WARN: Auftreten einer unerwarteten Situation

ERROR: Fehler

FATAL: Kritischer Fehler

9.3.4 Metrikanalyse

Die Metriken werden nach jedem Sprint protokolliert und Analysiert. Die Metriken helfen während einem Code-Review auf problematische Code Stücke hinzuweisen. Sie können auch beim Debuggen hilfreich sein.

9.3.5 Code Style Guidelines

Editor Standard

9.4 Testen

9.5 Modultest

Die Modultests werden, wie schon beschrieben ,während der Entwicklung mit Test-Driven Development erstellt. Zu jedem Modul werden Tests geschrieben, um eine möglichst hohe Code Coverage zu erreichen. Die Tests werden nochmals auf dem Build Server ausgeführt

um “it works on my machine” abzudecken.

Produkte:

- Irgend eine Unit-tester (je nach Sprache)
- Travis

9.6 Integrationstest

Bei den Integrationstests kommt eine Kombination aus automatisierten Tests, sowie eine Analyse des Ablaufs zum Einsatz. Die Tests werden an verschiedenen Schichten angesetzt (Presentation und Business-Layer). Die Unteren Schichten, die für den Integrationstest nicht relevant sind werden simuliert. Nachdem die automatische Tests durchgelaufen sind, werden die Log-Daten ausgewertet und mit den Sequenzdiagrammen abgeglichen. Es ist wichtig, das die Komponenten korrekt miteinander kommunizieren.

Produkte:

- Irgend ein Unit-tester (je nach Sprache)
- Für Presentation-Layer kommt <http://karma-runner.github.io/0.13/index.html> Karma zum Einsatz.
- Log-Daten

9.7 Systemtest

Die Systemtests richten sich nach den Use Cases. Auch hier kann mit Karma getestet werden (Das wäre dann eher ein sehr grosser Integrationstest). Die Use Cases werden von einem Tester nach Protokollangaben durchgeführt und dokumentiert. Auch hier können Log-Daten eine Hilfe für die Analyse sein.

Systemtests werden nach jedem Sprint durchgeführt um die geplante Funktionalität zu testen und so die neuste Version der Software releasen zu können.

Produkte:

- Karma
- Protokoll
- Log-Daten

9.8 Abnahmetest

Der Abnahmetest wird vom Kunden selbst durchgeführt.