



UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE MONITOREO DE
INFRAESTRUCTURA PARA EL CENTRO TECNOLÓGICO UCAMPUS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

EMILIO DAVID ABURTO LABRA

PROFESOR GUÍA:
WILLY MAIKOWSKI CORREA

MIEMBROS DE LA COMISIÓN:
SANDRA DE LA FUENTE GONZÁLEZ
MARÍA CECILIA BASTARRICA PIÑEYRO

SANTIAGO DE CHILE
2023

Resumen

El Centro Tecnológico Ucampus, es un centro de la Universidad de Chile que apoya la gestión de variadas instituciones de educación superior del país a través de plataformas web centralizadas en un sistema conocido como Ucampus. Para el desarrollo y mantención de este sistema se requiere de infraestructura y servicios tecnológicos que estén en constante monitoreo y funcionamiento, ya que caídas o incidencias sobre estos tienen graves implicancias en las instituciones.

Inicialmente, el monitoreo de la infraestructura y de los servicios subyacentes se realiza de manera desacoplada, con variados scripts que responden individualmente por cada uno de ellos. El único sistema que centraliza esfuerzos de registro es Nagios, herramienta que posee un registro instantáneo mas no histórico, elemento relevante al intentar investigar sobre los problemas. Por otra parte, las incidencias son tratadas a través de chats y una herramienta desarrollada por el mismo equipo del Centro Ucampus conocida como *CRM* que brinda una buena cobertura de la atención de usuarios y clientes. Aun así, existe deficiencia en la disponibilidad de información ante distintos usuarios incluso externos que requieren una vista pública.

Para abordar las problemáticas anteriores se decidió implementar nuevas herramientas que apoyen el monitoreo pero se descartó extender el *CRM*. Para ello, se complementó la herramienta Nagios con el sistema Prometheus y Grafana, herramientas elegidas para poder cubrir la falta de registro histórico y poder visualizar la información. Además, se diseñó e implementó un módulo que, a través de las APIs de Prometheus, pudiera brindar una categorización y un modelo lógico de “Institución”, “Servicio”, “Infraestructura” y “Test” respondiendo a los distintos usuarios.

Este flujo y módulo final se validó en dos ocasiones con los principales usuarios y clientes del sistema, el equipo de desarrollo de Ucampus. Si bien aprobaron y quedaron conformes con lo desarrollado, brindaron retroalimentación útil, donde algunas pudieron ser desarrolladas pero otras quedaron para etapas venideras del proyecto.

En conclusión, se lograron la mayoría de objetivos propuestos de comprender, diseñar e implementar un sistema de monitoreo. Si bien la extensibilidad del sistema permite la generación de una vista pública quedó pendiente el paso a producción y publicación definitiva de este.

A mi familia, mis amigos y al Centro Ucampus.

Tabla de Contenido

1. Introducción	1
1.1. Contexto y Oportunidad	1
1.2. Trabajo a realizar	2
1.3. Objetivos	3
1.3.1. Objetivo General	3
1.3.2. Objetivos Específicos	3
1.4. Estructura del documento	3
2. Estado del Arte	4
2.1. Monitoreo y Visualización	5
2.1.1. Nagios	6
2.1.2. Prometheus	7
2.1.3. Netdata	8
2.1.4. Elastic Stack	9
2.1.5. Otros	9
2.1.6. Resumen	10
2.2. CRM y Mensajería	10
2.3. Páginas Públicas	12
2.4. Ucampus	12
3. Situación Actual	14
3.1. Monitoreo	15

3.1.1. Infraestructura	15
3.1.2. Nagios y métricas de monitoreo	17
3.1.3. Nuevos Tests	18
3.2. Incidencias	18
3.2.1. Alarmas	19
3.2.2. Análisis y Post-Mortem	22
4. Análisis y Diseño	24
4.1. Discusión	24
4.2. Diseño	26
5. Implementación	29
5.1. Comunicación entre Nagios y Prometheus	29
5.2. Integración de Prometheus a Ucampus	30
5.3. Módulo de Ucampus	32
5.4. Validación	39
5.4.1. Revisión de Problemas	39
5.4.2. Validación con Equipo	41
6. Conclusión	43
6.1. Trabajo Futuro	44
Bibliografía	46

Índice de Ilustraciones

2.1. Arquitectura de funcionamiento de Nagios	6
2.2. Interfaz web de servicios de Nagios, mostrando el estado de un listado de ellos	7
2.3. Arquitectura de funcionamiento e integración de Prometheus	8
2.4. Vista general del sistema en Netdata	9
2.5. Vista general de un dashboard en Kibana	10
3.1. Infraestructura del Centro Ucampus y flujo de datos	16
3.2. Formulario de Contacto	19
3.3. Sistema de CRM	20
3.4. Mail con error de sistema	21
3.5. Publicación en Instagram sobre un error en la plataforma U-Cursos	21
4.1. Canal de notificaciones de Status de Ucampus en Telegram	26
4.2. Diagrama de solución propuesta de cómo se integra el sistema de Nagios con Prometheus y un módulo a desarrollar.	27
4.3. Estructura de base de datos para la categorización en servicio e institución a los distintos tests de Nagios	28
5.1. Estructura de funcionamiento de Iapetos	30
5.2. Ejemplo de respuesta a una consulta del número de clicks durante el último minuto en la institución “ucursos” en Prometheus	31
5.3. Vista inicial del módulo desarrollado, mostrando el estado general de las instituciones	34
5.4. Vista detallada de una institución en el módulo desarrollado	35

5.5. Vista de una institución con detalle de un servicio en particular	35
5.6. Vista de un servicio en todas las instituciones	36
5.7. Vista de una máquina o hostname en particular	36
5.8. Vista de configuración	37
5.9. Dashboard de Grafana para tests de Nagios	37
5.10. Dashboard de Grafana para visualizar clicks de usuarios en cada institución .	38
5.11. Dashboard de Grafana para visualizar la cantidad de usuarios conectados en cada institución	38
5.12. Interfaz web de herramientas para sysadmin de la plataforma Ucampus . . .	40
5.13. Interfaz web de vista en vivo de los clicks realizados en la plataforma Ucampus	41

Capítulo 1

Introducción

1.1. Contexto y Oportunidad

Ucampus se define en su sitio web como un Centro Tecnológico que desarrolla plataformas de apoyo a la gestión, automatizando los procesos de Instituciones de Educación Superior [2]. Es un organismo dependiente de la Facultad de Ciencias Físicas y Matemáticas (FCFM) de la Universidad de Chile y actualmente provee servicios a 8 instituciones, entre ellas la Universidad de Chile, la Universidad de O'Higgins, la Universidad Metropolitana de Ciencias de la Educación (UMCE) y la Universidad de Aysén, sumando más de 200 mil estudiantes sin considerar todos los funcionarios y profesores, que a su vez también son usuarios del sistema.

En la Universidad de Chile, por razones históricas se poseen dos plataformas: U-Cursos y Ucampus. La primera orientada al apoyo a la docencia y la segunda, a la gestión de procesos curriculares de la institución. En las demás universidades existe una única plataforma que unifica ambos servicios y es llamada Ucampus. En cuanto al software utilizado para proveer el servicio web responde a un “stack” clásico de desarrollo web conocido como LAMP[8] que consta principalmente de un servidor Apache que ejecuta código PHP y que realiza consultas a una base de datos MariaDB, todo esto montado sobre un sistema operativo GNU/Linux, en este caso CentOS.

Respecto a la infraestructura física, esta se divide, a grandes rasgos, en dos partes: una provee servicios para la Universidad de Chile y la otra provee servicios a las demás instituciones. En total se tienen aproximadamente 100 dispositivos, en su mayoría servidores, tanto físicos como virtuales, que cumplen funciones específicas dentro del sistema y se categorizan de acuerdo a su uso en las siguientes categorías: Servidores web con balanceo de carga, servidores de Bases de Datos, de almacenamiento, de respaldos, de servicios (DNS, email, etc.) y servidores de desarrollo.

Para asegurar el correcto funcionamiento y maximizar la disponibilidad de las plataformas antes mencionadas, la plataforma está programada para enviar emails con información útil al equipo de desarrollo cada vez que se encuentra un error grave en su ejecución. Además, se utiliza un software de monitoreo llamado Nagios[10] que notifica también mediante emails las anomalías físicas o de software de terceros encontradas en los múltiples servidores de

la infraestructura del centro. En ambos casos, la información de los emails es valiosa desde el punto de vista de un administrador de sistemas o del desarrollador, por separados, pero no refleja directamente el estado general de las plataformas ni su rendimiento durante el funcionamiento normal. Existe por lo tanto una carencia de información sintetizada que muestre una perspectiva global del estado de los servicios.

Además, tanto los emails enviados por la plataforma, como la información de anomalías existente en Nagios están diseñados para notificar cuando suceden problemas y mantener un registro histórico muy limitado de dichos eventos. Sin embargo, carece de un registro de indicadores que permitan identificar las causas subyacentes del problema, por lo que solo permite reaccionar ante eventos de falla. Al no estudiar el comportamiento de los servidores durante el uso normal del día a día, o en los instantes previos a un problema, se puede perder información relevante a la hora de estudiar las causas de un incidente y cómo evitarlo en un futuro.

Existe por lo tanto una necesidad de abstraer la infraestructura física subyacente a cada una de las plataformas y proveer al equipo de información tanto en tiempo real como histórica sobre cada instancia de Ucampus, de manera que se pueda comprender, por ejemplo, cómo se verá afectada cada institución cuando suceda una anomalía física.

1.2. Trabajo a realizar

Respecto al reporte y manejo de incidencias, actualmente sólo se reciben alertas sociales, originadas desde los usuarios de la plataforma que notifican manualmente cuando encuentran algún problema. Estas notificaciones llegan a un sistema de CRM, en el que se gestiona la comunicación con el usuario y resolución de ellas, en caso de ser problemáticas reales y relevantes de la plataforma. Sin embargo, este flujo no está bien definido en el caso de las incidencias espontáneas, en donde el equipo se coordina manualmente y solo vía medios de comunicación como Telegram[17] acerca del problema y quién será el responsable de solucionarlo, no existiendo un registro formal de la aparición, gestión y posterior resolución del incidente.

Es por estas razones que se propone la creación de un sistema de monitoreo de la infraestructura del Centro Ucampus, que incluya un registro histórico de métricas asociadas al hardware y software de las plataformas, de manera que se pueda cuantificar el estado normal de la plataforma y estudiar de mejor manera las situaciones anormales que puedan suceder. Se propone incluir también un sistema públicamente accesible que permita a los usuarios visualizar el estado general de disponibilidad de las plataformas, de forma tal que en el caso que no puedan acceder a ellas, puedan chequear personalmente si el problema es propio o es una incidencia en la plataforma.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo general de este trabajo es comprender, diseñar e implementar una herramienta que aborde las problemáticas asociadas al monitoreo de la infraestructura existente en el Centro Ucampus de manera tal que unifique los esfuerzos actuales y provea una perspectiva global del estado de los servicios.

1.3.2. Objetivos Específicos

1. Comprender el estado actual del Centro Ucampus respecto a infraestructura, herramientas de monitoreo y manejo de incidencias, identificando sus necesidades
2. Comprender cómo se aborda en la industria la problemática asociada al monitoreo de infraestructura y manejo de incidencias
3. Diseñar un prototipo web de una solución que se ajuste a las necesidades identificadas
4. Diseñar una arquitectura que provea de almacenamiento, recursos y disponibilidad suficientes para la solución modelada
5. Implementar en un desarrollo web la solución diseñada
6. Validar con el equipo de desarrollo del Centro Ucampus que la solución responda a las necesidades identificadas.

1.4. Estructura del documento

La siguiente memoria se estructurará con el capítulo 2 que consiste en una sección inicial de *Estado del Arte*, en donde se darán a conocer las herramientas estudiadas para abordar el problema. A continuación, en el capítulo 3, se abordará todo el estudio del estado actual del Centro Ucampus para ser analizado y diseñar la solución en el capítulo 4. Luego, se detallará la implementación desarrollada y su validación en el capítulo 5 para finalizar con los resultados y conclusiones del trabajo en el capítulo 6.

Capítulo 2

Estado del Arte

Para investigar sobre el estado del arte en términos tanto de herramientas disponibles como de la adopción de su uso en la industria, se conversó respecto al tema con profesionales egresados de Ingeniería en Computación que llevan algunos años trabajando en la industria sobre las prácticas y herramientas usadas en sus trabajos. Si bien se describirán concisamente estos sistemas, se profundizará con mayores detalles en las siguientes secciones.

En uno de los casos se conversó con un profesional del área de “Riesgos” en una empresa startup de Fintech, es decir, una empresa del mercado financiero que utiliza tecnologías para automatizar su funcionamiento y facilitar la gestión que sus clientes tienen de sus productos en el mercado bursátil. Esta empresa, desde sus inicios, ante la necesidad de resolver sus requerimientos de infraestructura decidió tercerizarla y preferir el modelo de Software como Servicio (SaaS) en donde su principal proveedor de servicios es Heroku, una plataforma de aplicaciones en la nube caracterizada por proveer un sistema de contenedores, las cuales pueden integrarse a muchos otros sistemas propios, entre ellos los de monitoreo y alerta. Dichas alertas automáticas se reciben luego en canales del software Slack, en donde los miembros del equipo pueden revisar, comentar y gestionar su resolución. Para incidencias de código, lo resuelven mediante el uso de la plataforma Github y su manejo de “Issues” integrado. Para su solución de página de status interna utilizan también un servicio SaaS que provee la empresa Atlassian, con alertas automatizadas vía Slack y vía Email.

Sobre esta experiencia de startup en el área financiera destaca una tendencia de los nuevos emprendimientos a tercerizar sus infraestructuras para crecer rápidamente con un equipo técnico pequeño, aunque en este caso la dependencia a una plataforma en particular los mantiene cautivos en Heroku, haciendo el costo en recursos de cambiarse a otra plataforma prohibitivo en términos de tiempo debido a la falta de interoperabilidad con otras plataformas que proveen servicios en la nube. Si bien hoy en día existen estándares de almacenamiento web y containerización en la nube que permiten mover con mayor facilidad la infraestructura de un proveedor a otro, siempre será otro factor a tener en cuenta a la hora de considerar el compromiso entre lo que ofrece la infraestructura propia versus la alternativa SaaS.

Otra de las empresas consultadas fue una consultora preocupada de realizar proyectos tecnológicos de distinta índole pero mayormente apoyando instituciones estatales. Sus procesos de desarrollo se adecuan a la metodología ágil y tanto el monitoreo y manejo de incidencias

lo poseen categorizado en niveles los cuales son acordados con los clientes para los distintos proyectos. Aun así, al ser mayormente concursos públicos deben adecuarse a estándares de aseguramiento de la calidad como lo es la norma *ISO 9001*. A grandes rasgos poseen tres niveles, un primer nivel que es abordado por correo donde se busca resolver consultas o aspectos principalmente superficiales de los sistemas, un segundo nivel donde abordan los errores de bases de datos, aplicación o infraestructura, y un último tercer nivel donde se manejan los errores de código usualmente críticos los cuales pueden ser manejados de una manera parecida a los del segundo nivel u otros que requieren de mayor urgencia y son considerados como “modificaciones en caliente” (*hotfix*). Tanto el segundo como tercer nivel son apoyados por la herramienta Jira Software donde poseen la gestión de los periodos de desarrollo de funcionalidades conocidos como *sprint*.

Sobre esta experiencia se destaca el nombramiento de un estándar como lo es la norma ISO 9001, la cual consiste en una norma definida por la *Organización Internacional para la Estandarización* que habla sobre aspectos como la gestión de la documentación, responsabilidad de los equipos, la gestión de los recursos humanos, infraestructura y ambiente de trabajo, como también la medición, análisis y mejora proactiva que debe comprometerse [6]. Por otra parte, un elemento relevante es la categorización en niveles y la utilización del software *Jira* [7], el cual consiste, según su página web, en una herramienta de apoyo al desarrollo de software adecuada para equipos que persiguen la metodología ágil y que permite planificar tanto incidencias como tareas, supervisar los tiempos generando métricas relacionadas a su resolución y apoyar esta gestión con informes y automatizaciones.

Por último, otra de las experiencia que se posee es del Ministerio de Cultura, la cual delega el manejo completo a una empresa externa que provee este tipo de servicio. En este caso *Entel*, reconocida empresa de telecomunicaciones de Chile, ofrece el servicio de mesa de ayuda, monitoreo de infraestructura e incluso desarrollo de cambios menores para asegurar el funcionamiento de los desarrollos que posee el ministerio. Además, coordina en caso de incidencias de mayor consideración con los proveedores externos correspondientes. Por otra parte, se menciona que si bien se posee lo anteriormente descrito, aún así existen esfuerzos de mantener un monitoreo interno proactivo de los servicios debido a que se ha descubierto un manejo más bien reactivo no histórico de la empresa externa, tal cual como se realiza en Ucampus actualmente y que se profundizará en el capítulo 3. Se declara que estos esfuerzos son básicos y abordados con desarrollos locales sin una mayor herramienta que apoye estos procesos.

Más allá de la experiencia localizada de cada una de estas historias se hace relevante la categorización de tres secciones importantes dentro del manejo que cada uno brinda: el monitoreo y visualización de información interna, la atención de usuarios, y como se resumen y comparten estos datos en sitios públicos. Esta categorización guía la profundización de la investigación de opciones en las siguientes secciones.

2.1. Monitoreo y Visualización

En esta sección se hablará acerca del software disponible para monitoreo y visualización de datos para métricas de infraestructura de hardware y software.

2.1.1. Nagios

Nagios [10] es un sistema autohosteable de monitorización de software y hardware muy utilizado en la industria. Fue creado por Ethan Galstad y lanzado en 1999. Es un producto de software libre, licenciado bajo la licencia GPLv2, gratuito y de libre descarga. Entre sus capacidades se encuentran la monitorización de servicios en red como SMTP, HTTP y SNMP, la monitorización de recursos de hardware como carga de procesador, uso de discos, memoria, estado de puertos, entre otros, posibilitando incluso monitorización remota via SSH o túneles SSL cifrados.

La instalación de Nagios es relativamente simple en la mayoría de las distribuciones de Linux, en las que ya existen paquetes preparados en los repositorios, aunque para utilizar la interfaz web se requiere instalar y configurar además un servidor web Apache.

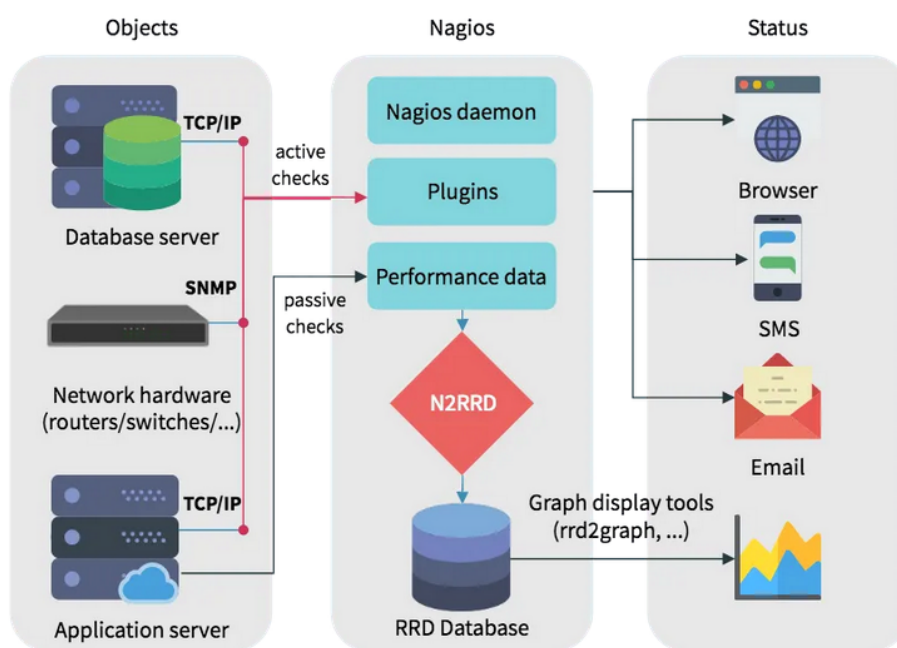


Figura 2.1: Arquitectura de funcionamiento de Nagios

El funcionamiento de Nagios (Figura 2.1) consiste principalmente en la ejecución de chequeos predefinidos o personalizados que monitorean el estado de algún sistema físico o de software. Estos chequeos son en general scripts que corren de forma automatizada y periódica. Dependiendo del resultado de estos, el sistema puede activar alertas a los administradores de sistemas previamente configurados.

Las métricas monitoreadas por Nagios son almacenadas en una base de datos RRD (Round Robin Database), en donde son rotadas regularmente, estando almacenadas sólo las del último período.

El sistema de alertas es personalizable, pudiendo notificar vía email o SMS, a través de la interfaz web u otro sistema instalable mediante plugins.

Además de las capacidades soportadas por defecto, Nagios posee una comunidad muy

grande de desarrolladores que han creado miles de plugins para chequear el estado de servicios. Por ejemplo, a través de plugins es posible chequear el estado de bases de datos, de routers, switches, entre otros.

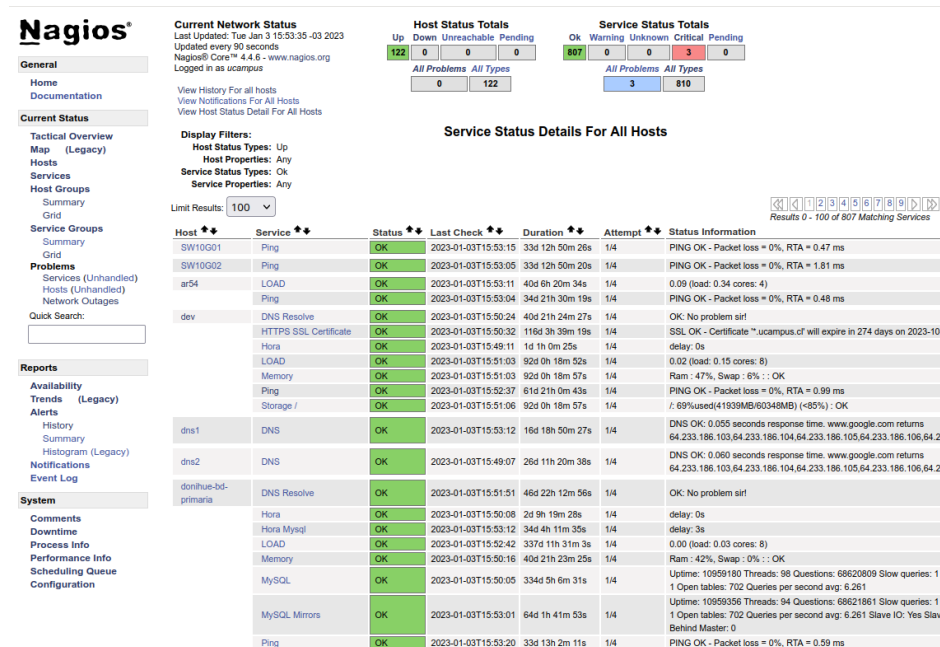


Figura 2.2: Interfaz web de servicios de Nagios, mostrando el estado de un listado de ellos

La interfaz mostrada en la Figura 2.2 es la visualización de servicios y hosts monitoreados de la interfaz web de Nagios. En ella se pueden observar un resumen de estado de hosts y de servicios en la parte superior, una tabla con el estado individualizado de cada servicio e información adicional sobre la fecha, duración e información obtenida durante el último chequeo.

2.1.2. Prometheus

Prometheus [13] es un software autohosteable de base de datos de series temporales especializado para el almacenamiento de métricas de sistemas de hardware y software. Fue desarrollado inicialmente dentro de la empresa SoundCloud en 2012, anunciado públicamente en 2015 y desde 2016 es un proyecto incubado por la Cloud Native Computing Foundation, representando el interés de grandes industrias tecnológicas como Google, Red Hat y DigitalOcean que lo utilizan en sus infraestructuras. Es un producto de software libre publicado bajo la licencia Apache 2.0, gratuito y de libre descarga.

Prometheus está ampliamente disponible para su instalación en los repositorios de las principales distribuciones de Linux. Además, cuenta con múltiples “exporters” que permiten obtener datos desde todo tipo de softwares y dispositivos, disponibilizándolos como métricas para Prometheus a través de una página de texto plano HTTP.

Entre sus capacidades destaca su posibilidad de almacenar métricas de forma altamente dimensional, permitiendo clasificarlas según un conjunto de llaves y valores, lo que es

beneficioso a la hora de filtrarlas en consultas utilizando PromQL, su propio lenguaje de consulta, que permite recuperar valores de las métricas y realizar operaciones matemáticas y estadísticas sobre ellas. Tiene también integración con una herramienta de visualización llamada Grafana [4] que permite crear dashboards que muestren consultas PromQL en visualizaciones como gráficos.

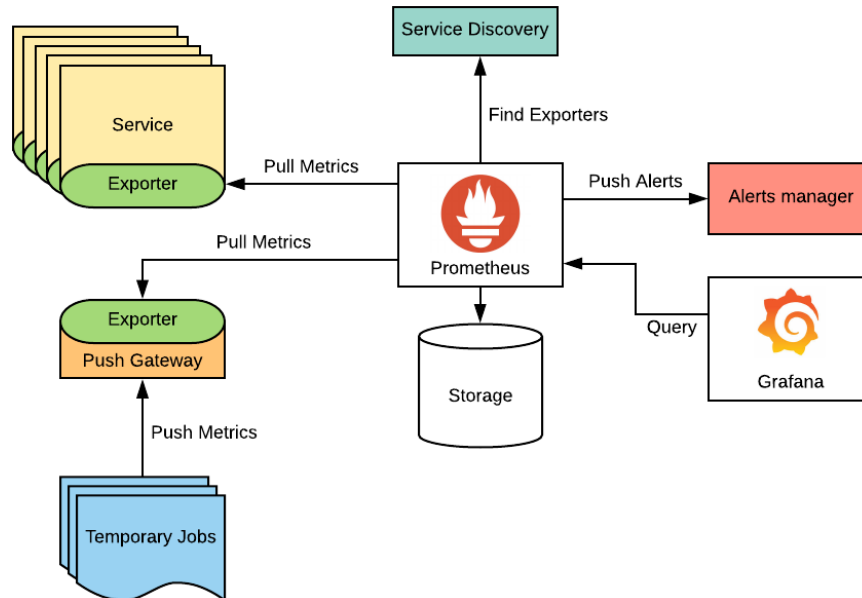


Figura 2.3: Arquitectura de funcionamiento e integración de Prometheus

Como se muestra en la figura 2.3, el funcionamiento de Prometheus se basa en: Primero, descubrir servicios a través de sus archivos de configuración; luego, consultar las métricas de los servicios encontrados (a través de exporters); almacenar las métricas obtenidas localmente y finalmente, recibir consultas PromQL desde otros servicios como Grafana, para que sean visualizados por los usuarios. Además, Prometheus cuenta con un servicio de alertas, mediante el cual puede notificar por email u otros servicios, cuando ciertas métricas se encuentren fuera del rango normal. También posee un servicio, llamado Push Gateway en el que servicios temporales pueden depositar métricas sin necesidad de quedarse esperando que Prometheus les consulte sobre ellas.

2.1.3. Netdata

Además, se investigó sobre otras soluciones de monitoreo, como Netdata [11], un software autohosteable que viene configurado por defecto para monitorear automáticamente miles de métricas de la máquina en que se instala, su software y hardware. Provee una interfaz web (figura 2.4) para visualizar y navegar las métricas y un sistema de alertas preconfigurado. Es un software libre, disponible gratuitamente y de libre descarga, con licencia GPLv3.

Los creadores de Netdata, sin embargo, han incentivado fuertemente la migración hacia su servicio de monitoreo en la nube. Para usarlo localmente y mantener un registro histórico

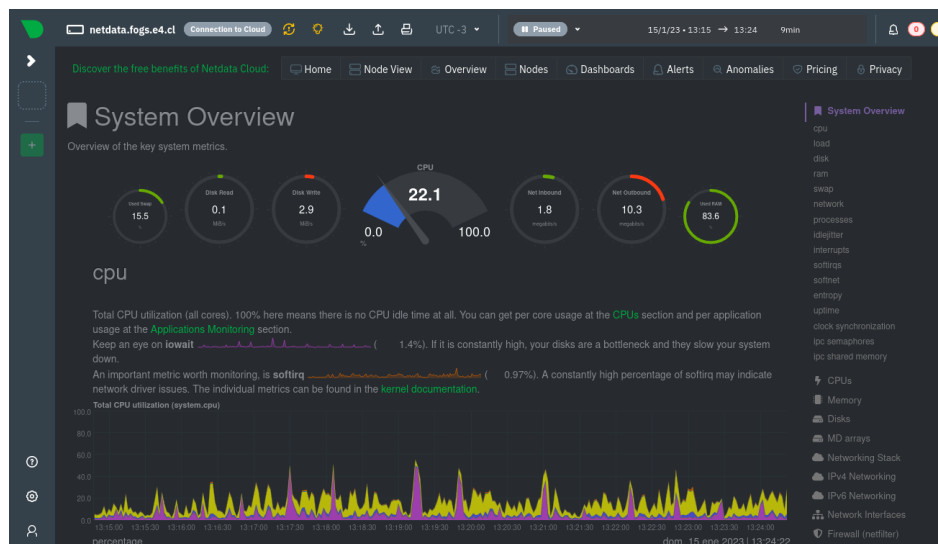


Figura 2.4: Vista general del sistema en Netdata

de las métricas de esta herramienta, se debe utilizar una base de datos para series temporales como Prometheus. Además, en términos de extensibilidad, si bien monitorea muchas métricas automáticamente, no es fácil integrar nuevos servicios ni configurar los ya existentes.

Considerando que el centro requiere un sistema extensible dado que a medida que se integran nuevos clientes, es necesario agregar los servicios de una nueva institución. Entonces Netdata no tiene las características necesarias para resolver este problema.

2.1.4. Elastic Stack

Elastic Stack, también conocido como ELK [3] es un conjunto de software autohosteables que comprende Elasticsearch, Logstash y Kibana para realizar tareas de almacenamiento de métricas, logs y su posterior visualización y consulta. Es un stack de software con licencia privativa, gratuito y de libre descarga. Sin embargo, no está diseñado específicamente para monitoreo, sino que es una herramienta de uso más general que algunos han decidido utilizar con este objetivo. Por lo tanto sus decisiones de diseño no están orientadas específicamente para la monitorización de infraestructura como tal, sino más bien como herramienta de uso general en la cual se pueden almacenar métricas.

Entonces, si bien Elastic Stack ofrece extensibilidad para monitorear los servicios necesarios por el centro y provee almacenamiento histórico de métricas, no provee soporte suficiente de la comunidad, lo que dificultaría la implementación de scripts de monitoreo en los servicios utilizados.

2.1.5. Otros

Si bien se analizaron otros sistemas de monitoreo y visualización (como New Relic, Splunk, entre otros), se observó que son en general herramientas de pago, o con planes gratuitos

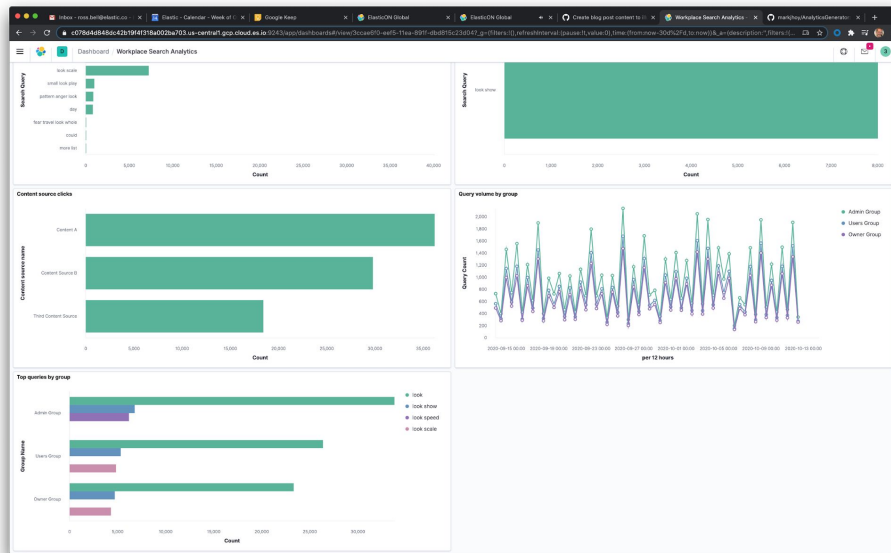


Figura 2.5: Vista general de un dashboard en Kibana

insuficientes para la escala requerida. Además, estas herramientas también son dependientes de la nube, elemento que no fue observado positivamente desde un inicio por el equipo del Centro.

2.1.6. Resumen

La tabla 2.1 presenta un resumen de las herramientas analizadas y la comparación entre sus características.

Tabla 2.1: Tabla comparativa de herramientas de monitoreo

	Costo	Servicio o Autohosteado	Soporte	Extensibilidad	Histórico
Nagios	No	Autohosteado	Suficiente	Sí	No
Prometheus	No	Autohosteado	Suficiente	Sí	Sí
Netdata	No	Autohosteado	Insuficiente	Sí	No
Elastic Stack	No	Autohosteado	Insuficiente	Sí	Sí
Otros	Sí	Servicio	Insuficiente	No	Sí

2.2. CRM y Mensajería

Las siglas en inglés CRM son utilizadas para hablar sobre los sistemas de “Gestión de Relación con Clientes”, que consisten en soluciones tecnológicas para interactuar con los clientes y usuarios de los sistemas. Dentro de los líderes en el área, se destaca la empresa Salesforce [14], quienes declaran ser la empresa número uno en productos CRM, sin embargo,

dentro de sus definiciones y servicios ofrecidos en la nube, se denota un enfoque más orientado a un equipo de ventas, postventa y marketing, que a un equipo de desarrollo de software.

En general el sistema de CRM ofrecido por esta y otras compañías, considera herramientas para administrar información de los clientes, organizar los intercambios de emails de forma inteligente, simplificar tareas repetitivas en los contactos con clientes y permitir retener a clientes actuales que no han mantenido contacto últimamente o capturar nuevos clientes con emails personalizados.

En el caso de Odoo [12], quienes se declaran el CRM Open Source número uno en el mercado, tiene un enfoque similar, siendo las ventas, postventa y marketing sus principales objetivos, lo que ofrece poca utilidad para un Centro de desarrollo tecnológico de software como Ucampus.

El caso del CRM de Jira [7], es una herramienta ideal para este caso, diseñada para un equipo de desarrollo de software, que permite tanto la organización interna del equipo como la capacidad de recibir retroalimentación de los clientes y usuarios de los sistemas desarrollados. Para lograr esto, Jira tiene una interfaz de tablero de tareas, un sistema de carta Gantt, entre otras visualizaciones. Esto permite al equipo distribuir mejor los esfuerzos para atender las necesidades de los clientes.

Además, Jira permite mantener un registro de los contactos con los clientes y usuarios, de manera que el equipo pueda coordinar tareas para resolver las incidencias que estos notifiquen. Sin embargo, es un producto privativo, que se provee bajo un modelo SaaS, es decir, es un software en la nube por el que se paga suscripción, lo cual no es muy aceptado en el equipo, que prefiere hostear su software en infraestructura propia y de preferencia, software libre.

En cuanto a herramientas de mensajería existen múltiples alternativas, como Slack [15], Telegram [17] y Whatsapp [19]. Entre ellas existen diferencias menores, como el hecho de necesitar un teléfono para registrar Whatsapp y Telegram, aunque en este último se pueda cambiar el identificador de usuario por un alias; por otro lado, para Slack solo hace falta un email. Sin embargo, es tal la competencia en el área de la mensajería que las características generales se encuentran más bien “normalizadas”, ya que replican las funcionalidades entre ellas. Estas características comunes son el envío de mensajes de texto y de audio, responder a mensajes, etiquetar a otros usuarios, envío de archivos, chats grupales, llamadas de audio y video grupales, entre otras.

Un punto a destacar tanto de Slack como de Telegram es la capacidad de acceder a los servicios vía una API pública, de manera que se pueden programar “bots” para notificaciones en un grupo o para desencadenar acciones en otros sistemas. En el caso de Whatsapp dicha funcionalidad es pagada y se reserva más que nada a negocios que requieran notificar a sus usuarios o tener “chatbots” que sirvan de atención al cliente de forma automática.

Cada uno de estos software tiene su punto a favor. En el caso de Slack, está orientado a equipos de trabajo, sin embargo para acceder a las características necesarias por el equipo, se requiere una membresía mensual pagada. En el caso de Whatsapp no permite muchas características avanzadas ni automatizadas de forma gratuita, pero su cantidad de usuarios es la más grande de las mencionadas. Finalmente en el caso de Telegram, provee una gran cantidad de características de uso automatizado avanzado por medio de bots y APIs en su

segmento gratuito de servicio.

2.3. Páginas Públicas

En un sistema web hace falta verificar desde una ubicación, preferentemente remota, que las plataformas se puedan visitar y que cargan en un tiempo razonable. Es por esta razón y por la experiencia consultada, que el equipo propuso también como problema a resolver, la necesidad de tener un sitio de *status*, que muestre la disponibilidad de servicios y les notifique cuando se presenten caídas.

Durante la investigación sobre alternativas de software con las capacidades requeridas, y que fuesen autohosteables por el Centro, se presentan dos alternativas: Statping y Cachet.

Statping [16] es una herramienta web de fácil uso, que muestra el estado de servicios y aplicaciones, sin dependencias externas ya que compila en un ejecutable “statically linked”, lo que hace que su despliegue en infraestructura remota sea sumamente fácil. Sin embargo, el desarrollo de esta herramienta parece haberse estancado cerca del 2020, por lo que algunos bugs quedaron sin solución, incluyendo lentitud al acceder al sitio público de estado y que el sistema de notificaciones envía falsos positivos acerca de sistemas caídos.

Cachet [1] es una herramienta web que tiene las mismas funcionalidades básicas de Statping. Su instalación fue un poco más difícil, pero prometía facilidad de uso y además posee una API para controlar los estados de los servicios. Dado esto, se pueden automatizar los cambios de estos estados desde otros servidores remotos. Sin embargo, la presencia de bugs y el notable estancamiento del desarrollo a principios del 2021 hicieron que el uso de esta herramienta tampoco fuese una posibilidad.

2.4. Ucampus

Para la comprensión del funcionamiento en la actualidad y parte de la discusión sobre la solución propuesta de los capítulos siguientes, es relevante comprender acerca del Centro Tecnológico Ucampus ya sea de su equipo como del desarrollo de las plataformas.

El equipo de Ucampus se divide en distintos equipos especializados, éstas áreas son cinco pero las principales relacionadas con la operación de los sistemas se denominan: Transferencia Tecnológica (TT), Comunicaciones y Desarrollo.

El equipo de TT, compuesto por 6 personas, mantienen un contacto estrecho con los encargados de las plataformas en cada una de las instituciones, manteniéndolos al día sobre los cambios y novedades realizadas sobre los sistemas. Por ejemplo, antes de iniciar las inscripciones académicas, se deben realizar todos los procesos de requisito necesarios para ello, es decir, configurar en el sistema los cursos a dictar y los estudiantes habilitados, modelar correctamente los planes de estudio y gestionar el número de cupos. Por lo tanto existe una comunicación constante para que este elemento esencial del funcionamiento institucional se

realice de forma idónea.

El equipo de Comunicaciones, compuesto por 2 personas, un Periodista y una Diseñadora, se encargan de realizar una labor transversal en términos de cómo se ven y comunican las novedades del Centro. Por ejemplo, al desarrollar una nueva funcionalidad, los iconos y la disposición de los elementos son visados por la diseñadora y al enfrentarse a una incidencia grave, la respuesta institucional también es manejada por este equipo, de forma que se gestione de manera adecuada.

El equipo de Desarrollo, compuesto por 7 personas, se encarga de desarrollar no solo la plataforma sino también la infraestructura que la soporta, manteniendo al día los sistemas y un estándar de disponibilidad de los servicios que haga deseable usar las plataformas. Dentro de sus tareas se encuentra el monitoreo de infraestructura y la resolución de problemas técnicos.

Respecto a la forma de desarrollar el software de las plataformas, el equipo de desarrollo considera un flujo de trabajo en que cada desarrollador maneja su propia instalación de prueba de Ucampus en el servidor de desarrollo y testeo, en el cual se agregan nuevas características al software y/o se corrigen errores existentes en la plataforma, pudiendo probarlos con los datos reales en un entorno seguro y delimitado antes de pasar los cambios a producción. Luego, una vez que los cambios están correctamente registrados en el sistema de control de versiones, estos son revisados por otro miembro del equipo y luego sincronizados a las distintas plataformas.

El Centro, como se mencionó anteriormente, posee dos plataformas principales: *U-Cursos* y *Ucampus*. Ambas son sistemas modulares. Es decir, están compuestas por una división lógica de funcionalidades que en su conjunto conforman el sistema. Existe un módulo principal llamado KERNEL que define la lógica de administración de módulos y se encuentra generalizado para cualquier otro sistema que lo requiera. Además, parte de este módulo posee la personalización para las distintas instituciones que poseen los servicios de Ucampus.

Algunos ejemplos de módulos que brinda la herramienta de Ucampus son: la toma de ramos o inscripción académica, el catálogo de cursos, los boletines de los estudiantes, entre otros. Estos fragmentos lógicos pueden comunicarse con elementos centrales del KERNEL o poseer su propia estructura de datos por lo que pueden ser independientes entre sí.

Usualmente el equipo de Desarrollo, implementa estos módulos siguiendo un paradigma Modelo-Vista-Controlador (MVC), separando el código desarrollado en tres carpetas respectivamente: *include*, *template* y *web*.

Capítulo 3

Situación Actual

En este capítulo se caracterizará el estado actual del Centro Ucampus respecto al problema de monitoreo de infraestructura e incidencias. Para ello se necesita considerar que el Centro Ucampus provee de servicios de software a universidades y otros centros educativos mediante una plataforma llamada también Ucampus. Los servicios prestados por esta plataforma son de administración curricular por parte de la institución y también un software de apoyo a la docencia para profesores y alumnos.

En el Centro Ucampus no se tiene una manera única de responder a las situaciones anómalas, es decir, no existe una forma sistemática de llevar registro de las incidencias ni de coordinar la respuesta, en parte debido a que al ser un equipo de desarrollo pequeño, hasta el momento no ha sido estrictamente necesaria una coordinación mayor que la que provee un servicio de mensajería instantánea como Telegram. Así también, la respuesta y solución a los problemas que surgen son resueltos en base a experiencia del equipo de trabajo con problemas del pasado o con el conocimiento cabal de cómo funciona la plataforma, debiendo muchas veces mirar en detalle los archivos de *logs* dejados en los múltiples sistemas, con tal de encontrar la causa de la incidencia. Luego, para evitar que suceda el mismo problema en el futuro, muchas veces se desarrolla una forma de chequear que el sistema esté bien configurado y se automatiza dicho chequeo usando principalmente el software Nagios. Existen por lo tanto varios scripts que chequean la configuración de cada servidor, todo esto corriendo en una máquina que monitorea al resto de la infraestructura.

Considerando lo anterior y la comprensión actual de la situación, en este trabajo se propone, como ya se adelantó en la Introducción, dividir el problema en dos subproblemas: Monitoreo e Incidencias. En el área de Monitoreo, el problema que se busca resolver es el de tener información veraz y disponible sobre el estado de la infraestructura y el software que utilizan las plataformas de Ucampus. Por otro lado, el área de Incidencias busca resolver el problema de recopilar información relevante y coordinar la respuesta del equipo ante un evento anómalo, de forma que su resolución sea registrada.

Es importante notar que ambos subproblemas pueden ser considerados independientes el uno del otro, pero reconocer que son problemas complementarios permite diseñar la solución a uno de ellos pensando en una futura solución completa que integre ambos. Por ejemplo, parte del manejo de incidencias, en una solución óptima, contiene mediciones útiles de monitoreo

que informen las decisiones a tomar para corregirla correctamente.

En cuanto al acercamiento que se ha tomado en Ucampus al respecto, cabe notar que corresponde al monitoreo local de una infraestructura relativamente pequeña, ubicada físicamente en un único datacenter. Esto permite tener un control y acceso físico a cada una de las máquinas del sistema.

Se realiza monitoreo de esta infraestructura tanto a nivel de redes como de sistemas de forma automatizada, mediante el software Nagios principalmente. Además, de forma manual, se realizan chequeos de métricas de latencia y disponibilidad cuando se percibe lentitud en los servicios. A su vez, al desarrollar el software, los cambios de la plataforma son testeados con el fin de detectar problemas de carga o lentitud causados por estos cambios.

Ante una incidencia, las plataformas están configuradas para alertar de errores graves mediante emails, tras lo cual el equipo de desarrollo se coordina por mensajería instantánea para resolver el problema. Sin embargo, no hay registro histórico de las anomalías que suceden ni un proceso de *post-mortem* definido, es decir, luego de solucionada la incidencia no se ha establecido un protocolo para analizar lo sucedido y limitar la posibilidad de que suceda en el futuro.

A continuación se detallarán ambos subproblemas: de Monitoreo y de Incidencias, por separado.

3.1. Monitoreo

3.1.1. Infraestructura

Para describir el estado actual de monitoreo en la infraestructura del Centro Ucampus, hace falta detallar cómo se compone su infraestructura y el software que se utiliza para proveer la aplicación y comunicarse con los clientes web y de aplicación móvil.

A grandes rasgos, el sistema utiliza un *stack* de software denominado LAMP, constituido por un servidor con sistema operativo GNU/Linux (en este caso la distribución CentOS), Apache como software de servidor web, MariaDB como software de base de datos y PHP como lenguaje en que se implementa la aplicación web. Se utilizan además, Memcache como caché para optimizar la necesidad de consultas pesadas a la base de datos, OPCache para acelerar el acceso a los scripts de PHP, Manticore como software indexador y de búsqueda en los distintos buscadores de la plataforma, Redis (en su modo PubSub) para notificar y sincronizar cambios en la base de datos entre los múltiples servidores de backend, y NodeJS para gestionar dicha sincronización de base de datos, manejar las notificaciones push a los clientes web y de aplicación móvil, además de contabilizar la cantidad de clientes conectados, los clicks realizados en la plataforma y las urls visitadas, que se proveen vía una conexión SocketIO directamente desde cada cliente. Un diagrama de resumen de esta infraestructura y sus componentes se encuentra en la figura 3.1.

Cada instancia de las plataformas que provee el Centro Ucampus, dependiendo de la

cantidad de clientes simultáneos, puede ser atendida por más de un servidor con tal de balancear la carga de cada servidor de backend y evitar lentitud o falta de disponibilidad del servicio por sobrecarga de clientes. Con este fin, cada solicitud HTTP de la plataforma pasa por un servidor de entrada, denominado Balanceador en el diagrama, encargado de distribuir la carga entre varios servidores de backend, según se encuentren disponibles y con baja carga. Esto se realiza de acuerdo a informes de salud que envía cada servidor de backend al servidor de entrada, mediante un software interno llamado Ucampusd.

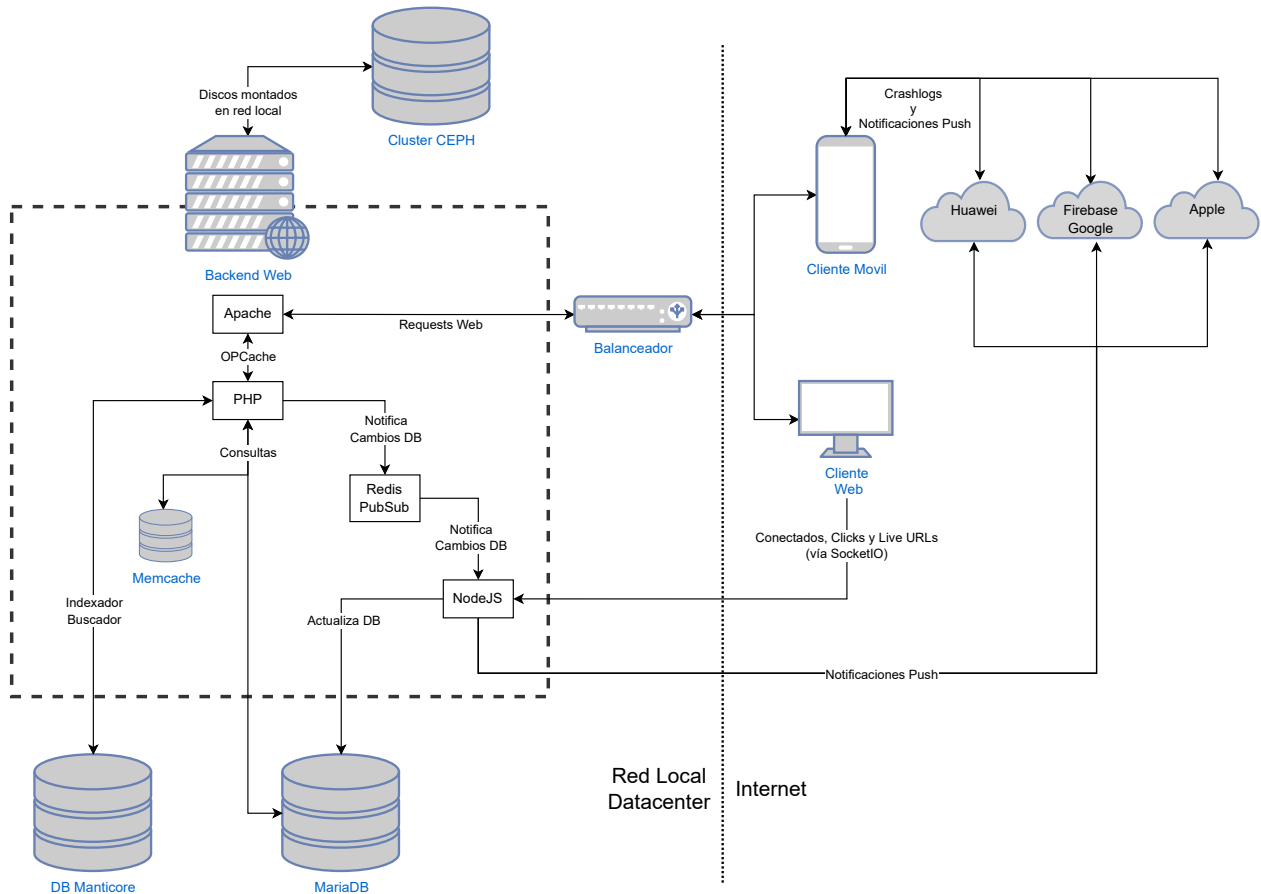


Figura 3.1: Infraestructura del Centro Ucampus y flujo de datos

Considerando el diagrama de infraestructura, de este sistema actualmente se monitorea:

- Instalación inicial: Existe un script de PHP encargado de revisar si las dependencias de software están instaladas y en qué versión se encuentran, chequear los parámetros de PHP necesarios y verificar que algunas opciones de Apache se encuentren correctamente configuradas.
- Carga y uptime de cada servidor de backend, principalmente para informar al servidor frontal Balanceador.
- Estado de OPCache, para verificar que se estén cacheando y acelerando el acceso a archivos de PHP

- Archivos de Logs: Accesos y errores de Apache, "Slow queries" de MariaDB, maillog del Servidor de Emails y logs de los múltiples procesos de NodeJS encargados de notificaciones Push para diferentes plataformas.
- Registro (en tiempo real) de la cantidad de Conectados a cada plataforma, Clicks realizados y URLs cargadas por cada usuario.
- Estado de Memcache: en donde se puede ver la utilización del caché y caducar entradas o forzar un recaché
- Estado de clúster de almacenamiento CEPH: en donde se puede ver información sobre la latencia de acceso al almacenamiento en red, porcentaje de utilización, entre otras métricas.
- Información sobre móviles: En cada proveedor (Apple, Google y Huawei) existen interfaces web que muestran información sobre los errores de la aplicación en cada plataforma, cantidad de usuarios, versiones, entre otras métricas. En este caso no corresponde a una infraestructura local, por lo que el monitoreo se realiza de forma esporádica y sólo manualmente.

Sólo para los archivos de logs se posee un registro de errores pasados. Además, para cada ítem del listado anterior existe un lugar en particular en el que revisar su estado actual. Sin embargo, la mayor parte de la monitorización se realiza utilizando el software Nagios, que detallaremos en la siguiente subsección.

3.1.2. Nagios y métricas de monitoreo

El software Nagios es una solución de software libre diseñado para monitorear servicios, aplicaciones y hardware de una infraestructura. Dicho monitoreo consiste en ejecutar de forma periódica tests en la máquina local o máquinas remotas que determinen el estado de un cierto componente del sistema monitoreado. Dependiendo del resultado de cada uno de estos tests y sobre todo si ha fallado múltiples veces, se pueden configurar alertas mediante email para que el equipo de desarrollo y administración de servidores pueda solucionar el problema.

En el caso del centro Ucampus, el software Nagios se utiliza para realizar chequeos de: resolución DNS, zona horaria correcta, carga, utilización de memoria RAM, estado de MariaDB, estado de discos de almacenamiento, estado de certificados web SSL, estado del servidor de emails, estado de Ucampusd en servidores workers, estado de los procesos de NodeJS, Memcache, Manticore y Redis, y chequeo de la realización de backups. Una vista de su interfaz mostrando servicios monitoreados se puede observar en la figura 2.2, de la sección Estado del Arte.

En total, utilizando Nagios se monitorean más de 800 indicadores, distribuidos en 122 dispositivos, tales como servidores físicos, servidores virtuales, routers y switches.

El software Nagios constituye entonces una de las principales formas de monitoreo de infraestructura que se utiliza en la actualidad, que se ha ido construyendo con los años,

utilizando configuraciones de monitoreo comunes (principalmente el uso de recursos como la CPU, memoria RAM y discos duros) y también a base de identificar problemas previamente no detectados que son más específicos del software utilizado en las plataformas de Ucampus, para los que se agrega un test y se configura para que se compruebe periódicamente en cada uno de los servidores que corresponda.

Dada la preponderancia de la herramienta Nagios, se ejemplificará y describirá a continuación el flujo de creación de nuevos *Tests* a través de esta herramienta.

3.1.3. Nuevos Tests

Para detallar el proceso de agregar nuevos tests al sistema de monitoreo actual, se utilizará como ejemplo una situación relativa a la validez de los certificados digitales SSL sucedida en los sitios web de las plataformas Ucampus.

Hasta hace unos meses, en todas las plataformas de Ucampus se utilizaban certificados digitales SSL proveídos por el sistema Let's Encrypt para servir los sitios web de forma segura utilizando HTTPS. Dichos certificados aseguran su validez mediante una cadena de confianza criptográfica, que finalmente verifica que un emisor de confianza del dispositivo haya autorizado su emisión. Los emisores de confianza del dispositivo se mantienen actualizados en cada dispositivo mediante las actualizaciones de sistema, lo cual produjo que en equipos de escritorio y móviles más antiguos, que ya no reciben actualizaciones de sistema, detectaran como inválido algunos certificados utilizados por las plataformas Ucampus.

Dicho problema no fue detectado a tiempo por el equipo de desarrollo, el cual se enteró mediante la notificación de algunos clientes que no podían acceder a los sitios web, ya que les aparecía que no eran sitios confiables al intentarlo.

Fue entonces que se investigó la posible causa y se optó por cambiar de proveedor de certificados SSL, además de buscar la forma de detectar en el futuro un error de estas características. Finalmente, luego de una investigación sobre cómo automatizar la detección, se agregó un script de testeo de certificados SSL inválidos o vencidos a la lista de tests existentes en Nagios.

Por lo tanto, el proceso mediante el que se agregan tests al sistema se divide básicamente en tres etapas: Identificación y aislamiento del problema a testear; investigación respecto a la automatización de su detección y finalmente, implementación de la solución elegida.

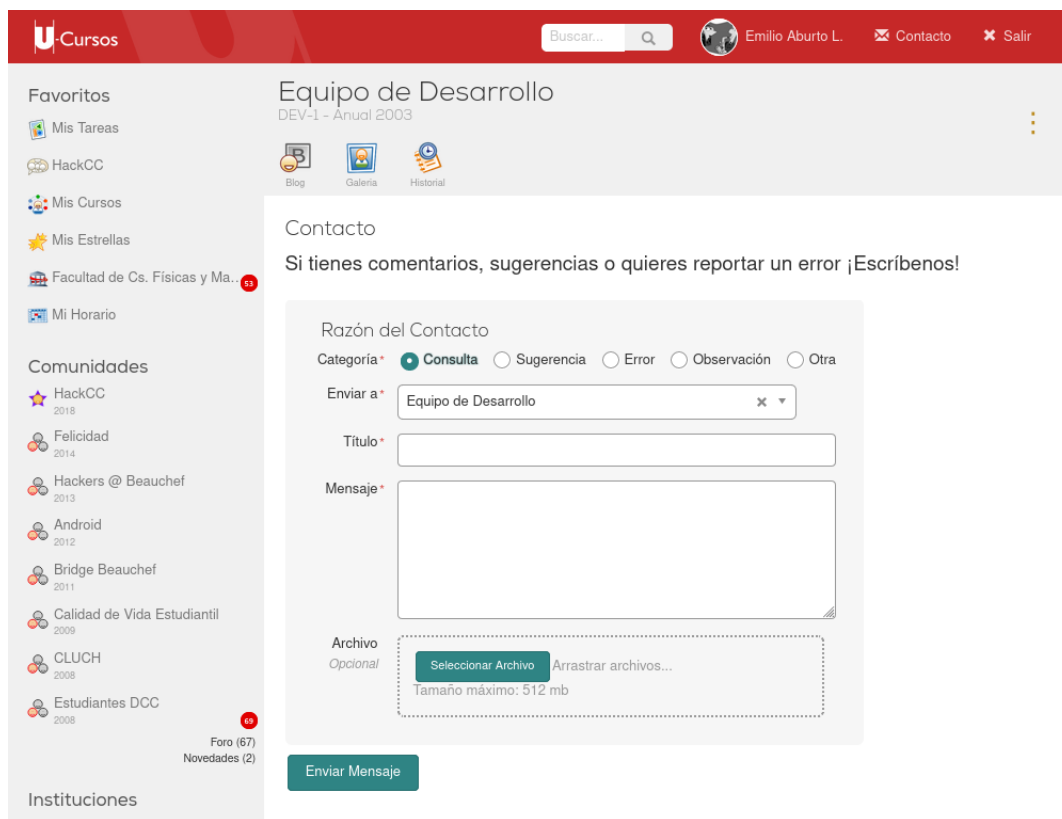
3.2. Incidencias

Las incidencias en el Centro Ucampus corresponden a sucesos que disminuyen la disponibilidad de las plataformas, impiden la realización de ciertas acciones de forma correcta en ellas o afectan la redundancia en algún componente de la infraestructura física. Se identificó que cada incidencia consta de tres etapas principalmente: Alarma, Análisis y Post-mortem.

3.2.1. Alarmas

En la etapa de alarma, se recibe un aviso por algún medio sobre un problema que concierne al funcionamiento de las plataformas. Existen actualmente alarmas informales, en donde usuarios del sistema contactan a personas del Centro para reportar problemas por vías externas, generalmente aplicaciones de mensajería instantánea; alarmas autogeneradas, en donde algún chequeo de salud de Nagios o algún error de la plataforma envía un email dando aviso del problema y sus detalles; y también un sistema de alarmas sociales formales, denominado CRM, mediante el que cualquier usuario puede escribir al equipo del Centro, sobre dudas o problemas respecto al sistema.

En el caso de las alertas recibidas mediante este sistema, son redactadas por los usuarios en un formulario de contacto como el mostrado en la figura 3.2. Estos mensajes son recibidos por el equipo del Centro en el sistema de CRM mostrado en la figura 3.3, desde donde puede ser asignado a quien pueda resolverlo, teniendo la posibilidad de intercambiar emails con la persona que reportó, escribir comentarios sobre el problema y marcarlo como resuelto cuando corresponda.



The image shows a web interface for 'U-Cursos' with a red header. The user is logged in as 'Emilio Aburto L.' and is viewing the 'Equipo de Desarrollo' page for 'DEV-1 - Anual 2003'. The page features a sidebar with navigation links like 'Favoritos', 'Comunidades', and 'Instituciones'. The main content area is titled 'Equipo de Desarrollo' and contains a 'Contacto' section with the heading 'Si tienes comentarios, sugerencias o quieres reportar un error ¡Escríbenos!'. The form includes a 'Razón del Contacto' section with radio buttons for 'Consulta' (selected), 'Sugerencia', 'Error', 'Observación', and 'Otra'. Below this is a dropdown menu for 'Enviar a' set to 'Equipo de Desarrollo', a 'Título' text field, and a 'Mensaje' text area. There is also an 'Archivo' section with a 'Seleccionar Archivo' button and a note 'Opcional' and 'Tamaño máximo: 512 mb'. A green 'Enviar Mensaje' button is at the bottom.

Figura 3.2: Formulario de Contacto

Este sistema de CRM se encuentra en una posición privilegiada desde el punto de vista de las comunicaciones con los usuarios de las plataformas y con el resto del equipo, ya que permite a todos en el Centro Ucampus revisar, asignarse o delegar situaciones, dependiendo del área que deba encargarse de resolverla. Para este fin, las solicitudes pueden ser ordenadas en la interfaz según quién sea el encargado, revisar cuales aún no han sido asignadas, además

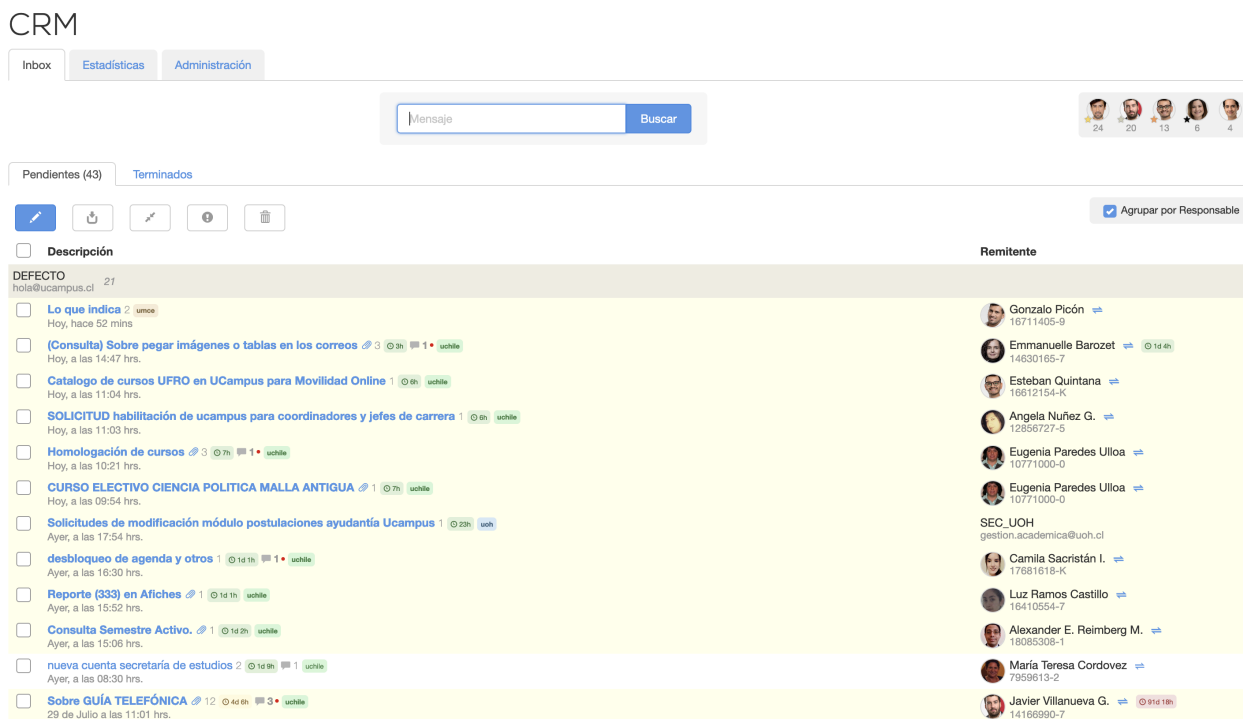


Figura 3.3: Sistema de CRM

de revisar quiénes ya han mirado el caso y cuando fue la última vez que lo vieron, lo cual informa a los demás tanto su carga de trabajo como si ya están enterados sobre alguna solicitud pendiente.

En cuanto a las alarmas autogeneradas por la plataforma y por Nagios, estas se envían automáticamente por los servidores del Centro a los emails de algunos trabajadores del área de desarrollo. En particular la plataforma avisa a los encargados del desarrollo de la aplicación, ya que los errores suelen ser de código o consultas SQL mal escritas, como se puede ver en el ejemplo de la figura 3.4, mientras que Nagios notifica a los administradores de sistema del equipo, que pueden revisar los problemas que surgen a nivel de Sistema Operativo de los servidores o en el hardware.

Respecto a alarmas sociales informales, estas suelen ser notificadas por personas usuarias del sistema por medios fuera de la plataforma (es decir, no mediante el CRM), por personas escribiendo algún tema en los foros de la plataforma o mediante redes sociales públicas como Twitter o Instagram, que es el caso de ejemplo de la imagen en la figura 3.5. En estas circunstancias, al ser alertados o al notar alguna de estas publicaciones, el equipo analiza su gravedad y relevancia con tal de ver si es algo que requiere resolverse o mejorarse de alguna manera. En el caso de la figura de ejemplo, parte del código utilizado para hacer debug de la aplicación quedó por error en la versión de producción y si bien no causaba que malfunctionara el sistema, mostraba información que no correspondía mostrar.

Luego de recibida la alarma y dependiendo de si es una alarma formal o informal, es posible que miembros del equipo compartan información al respecto en el grupo de Telegram de coordinación de desarrollo, en donde se conversa sobre la gravedad de la alerta, un análisis preliminar de afectados, posibles causas y también donde buscar mayor información para

3.2.2. Análisis y Post-Mortem

La próxima etapa luego de las alarmas es la comprensión y resolución del conflicto. Esta etapa es realizada mayormente por el área de desarrollo del Centro y se completa revisando todas las herramientas de monitoreo antes descritas en la sección *Monitoreo*.

Solo una vez que el problema ha sido resuelto, el equipo discute sobre acciones futuras para evitarlo, soluciones definitivas y también si se hubiese podido evitar el problema con algún chequeo faltante. En el intertanto, hace falta también comunicar al resto del equipo del Centro Ucampus sobre si la situación ha sido resuelta, para que se pueda responder adecuadamente a los clientes de las plataformas. Es común que al no ser la resolución de problemas un procedimiento establecido, este último paso sea olvidado y por lo tanto comunicacionalmente el problema aparentemente aún existe, aunque técnicamente ya se haya resuelto. A este proceso posterior a la resolución del conflicto se le denomina Post-mortem.

Dependiendo de la gravedad, duración y origen del problema, en algunas ocasiones ha sido necesario responder comunicacionalmente ante las instituciones por problemas de la plataforma o su conectividad. Para ello el equipo de Desarrollo se coordina con los equipos de Transferencia Tecnológica y de Comunicaciones. Entre ambos equipos conforman la presencia en redes sociales del Centro y la contraparte en comunicación más directa con los encargados de las instituciones que utilizan Ucampus, es decir se comunican tanto con nuestros usuarios como también nuestros clientes. Estos dos equipos utilizan entonces la explicación más técnica que podría proveer el equipo de desarrollo y la comunican a los afectados (de la forma que corresponda a cada uno) como forma de mantenerlos al tanto de lo sucedido y de las medidas tomadas o que se tomarán al respecto en el futuro.

Gran parte de esta coordinación entre personas del equipo sucede a través de conversaciones en varios chats grupales mediante la plataforma de mensajería instantánea Telegram, utilizada ampliamente por los trabajadores del Centro, sobretodo en un período de pandemia global en que el trabajo presencial no ha vuelto en su totalidad, por lo que parte del equipo está usualmente de forma remota. Actualmente se poseen cuatro grupos principales: Producción, Desarrollo, Implantación y Urgencias.

En el chat de Producción tienen acceso tres personas que cuentan con los accesos más amplios del centro, considerando los servidores de producción, registros históricos de errores y almacenamientos utilizados por todas las plataformas.

Al grupo de Desarrollo tienen acceso los siete miembros del equipo de desarrollo y se utiliza para coordinar tareas diarias y de incidencias.

Al grupo de Implantación tienen acceso miembros de Desarrollo y Transferencia Tecnológica y es donde se encargan de coordinar las nuevas funcionalidades o desde donde surgen las incidencias comunicadas por las instituciones. Además es el lugar de encuentro para abordar temas interdisciplinarios como las comunicaciones externas que no son puramente técnicas ni políticas sino una mezcla.

Por último, el grupo de Urgencias es un subconjunto de personas de las tres áreas mencionadas inicialmente, en el cual se discute la necesidad de responder comunicacionalmente

indicado en párrafos anteriores.

Debido a la anterior distribución de grupos y responsabilidades, se han detectado problemas de disponibilización de información para todo el equipo debido a que, por ejemplo, parte de la información para resolver conflictos de parte de los desarrolladores solo se discutían o son accesibles por el grupo de Producción, o que la creación de un post-mortem efectivo depende de la disponibilización de información dentro del chat de Urgencias, etc.

Capítulo 4

Análisis y Diseño

4.1. Discusión

En el equipo de desarrollo del Centro Ucampus se observan aspectos culturales y flujos de trabajo ya establecidos entre sus integrantes, que aunque no estén formalmente definidos, permiten al equipo organizarse de forma eficiente a la hora de enfrentar incidencias en las plataformas. Dado que no se observa la necesidad de modificar dichos procesos, en este trabajo se buscará crear herramientas y estructuras que apoyen y se integren a dichos flujos de trabajo siempre que sea posible.

En cuanto a la infraestructura física del Centro Ucampus, se observa que consiste en servidores físicos ubicados en un único datacenter, lo que reafirma la idea y cultura del equipo, que evita depender de servicios externos tercerizados y prefiere siempre hostear y desarrollar las herramientas que sean necesarias en la infraestructura propia. Por lo tanto, la solución a desarrollar será implementada en la infraestructura propia del Centro. Esta modalidad no necesariamente se observa en otras compañías discutidas en el Estado del arte, que prefieren tercerizar sus servicios.

En la solución actual de monitoreo existen registros de datos repartidos en múltiples archivos de logs e interfaces web, que no proveen una visión unificada del estado de los servicios, siendo el software Nagios el único lugar en que ha sido posible centralizar parte de los tests de estado de los sistemas.

En todas las soluciones de monitoreo desarrolladas hasta el momento, solo han sido consideradas métricas instantáneas del estado de los servicios. Es decir, no se tiene persistencia de los datos acerca de la disponibilidad de los sistemas de forma histórica. Por otra parte, ante un problema, la reacción del equipo es solucionarlo en la inmediatez, pero queda pendiente el realizar un análisis de lo sucedido. Para este análisis posterior nuevamente se vuelve necesario un registro histórico. Con este análisis del registro se pueden entonces crear nuevos tests que aceleren la detección del mismo problema en el futuro y simplificar la creación del post-mortem.

Como parte de los desafíos que se observan, la integración de la solución a desarrollar con

el flujo ya existente de creación de nuevos tests en Nagios, es primordial. Esto ya que, como mencionamos anteriormente, queremos mantener, integrarnos y mejorar los flujos existentes, sin agregar mayores sobrecargas a estos procesos debido a que podría ser contraproducente y provocar una falta de actualización requerida por estos sistemas de monitoreo.

Dado lo anterior, y la decisión de mantener Nagios, existen otros problemas a ser abordados en adición a la falta de registro histórico. Por una parte la visualización de la herramienta Nagios si bien es útil para un administrador de sistemas, no refleja de manera evidente una perspectiva global de los servicios utilizados por la plataforma ni una manera gráfica de evidenciar los distintos estados. Además, ante el fallo de un test dentro del sistema de Nagios, esto no refleja con claridad la institución o el servicio que se ve afectado, siendo esto una necesidad latente para el equipo de TT que usualmente no tiene conocimiento técnico y necesita disponer de información acerca de las instituciones.

Respecto al sistema de alarmas formales, el CRM existente en el Centro Ucampus es un desarrollo propio y por lo tanto, extensible en un futuro para albergar un sistema de Incidencias internas que se adecue a las necesidades del equipo, tales como creación automática de “Casos” al suceder una incidencia y visualización de información relevante de monitoreo de estos. Se evidencia que actualmente cubre las funcionalidades básicas de los sistemas del estado del arte discutidos pero las alternativas observadas no se adecuan ya sea por la naturaleza del equipo, su cultura o por los cambios requeridos para poder ajustarlos para un equipo pequeño. Por ejemplo, el sistema Jira considera procesos de desarrollo con muchos más pasos pensando en equipos de mayor tamaño o con una estructura de toma de decisiones con mayor verticalidad.

Por otra parte, las alarmas informales como publicaciones en redes sociales o mensajes directos a integrantes del equipo por mensajería instantánea, suelen declarar con mayor rapidez la existencia de problemas en las plataformas. Por esto mismo, por esta espontaneidad, se vuelven imprescindibles dichas alertas. Además, existe y se busca constantemente mantener la cercanía con los usuarios por parte del equipo, de manera que sientan la confianza de consultar y proponer nuevas funcionalidades. De esta manera es evidente para ellos reportar problemas de los sistemas y sería infactible eliminarlos o dirigirlos hacia una herramienta distinta.

Por lo mencionado en los dos últimos párrafos, en donde se demuestra que no existe la necesidad inmediata de realizar cambios en lo que respecta a las alarmas, se mantendrá el sistema de CRM desarrollado por Ucampus. Aun así, existen problemas con cómo informar a los usuarios luego de una incidencia y la creación de post-mortem que requieren de atención.

Si bien inicialmente el equipo de desarrollo propuso la necesidad de resolver el tema de una página de status y el tema de monitoreo como problemas independientes y como soluciones desacopladas, a lo largo de la investigación de las soluciones para informar a los usuarios luego de una incidencia, se fue haciendo cada vez más evidente que ambas herramientas responden a una necesidad más global de monitoreo de infraestructura.

Para solucionar dicha necesidad, se probaron los software Cachet y Statping. Al intentar instalar Cachet en una máquina de prueba, los pasos de instalación, así como el software proveído no estaban actualizados y se encontraron errores no documentados. Por otro lado,

la instalación de Statping funcionó bien y se dejó una página de status funcional inicialmente, que con el tiempo evidenció problemas de rendimiento y tiempos de carga muy largos en su interfaz web.

Posterior a las pruebas, se observó que los desarrolladores del software Statping y Cachet abandonaron su desarrollo. Si bien finalmente se despliega un sitio de status útil y capaz de proveer al equipo de información sobre el estado de los servicios y notificar vía Telegram cuando se cae alguno de los sistemas (figura 4.1), no se logró la aceptación requerida por falta de configuración y problemas de latencia, por lo que se decidió desarrollar esta herramienta dentro del sistema de monitoreo.

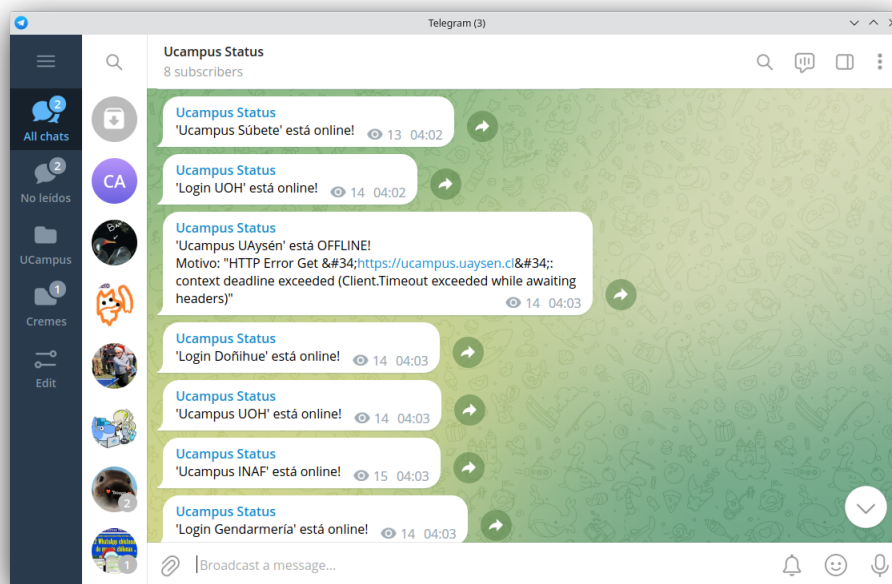


Figura 4.1: Canal de notificaciones de Status de Ucampus en Telegram

4.2. Diseño

Se propone entonces inicialmente complementar el sistema de Nagios con Prometheus, considerando que este último tiene registro histórico de métricas por un tiempo configurable, un sistema de alertas de monitoreo de dicha métricas e integraciones para monitorear sistemas y otros softwares que se utilizan en el centro Ucampus.

Si bien Prometheus y Grafana cubren la necesidad de registro histórico y la visualización de información, aún se requiere de una categorización de los tests que permita saber con claridad la institución y el servicio que se ve afectado. Por otra parte, se poseen distintos usuarios de distinto nivel técnico que observarán la información y, por consiguiente, se debe desarrollar una vista dedicada para cada uno de ellos.

Debido a esto, se propone construir un módulo, ayudado por el sistema de permisos y

administración que posee el KERNEL de Ucampus, para proveer de la categorización y distintas vistas por tipo de usuario requeridas.

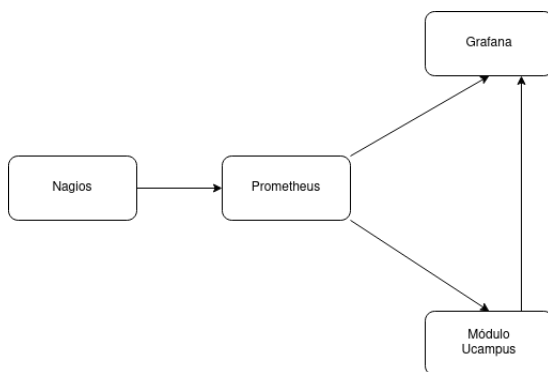


Figura 4.2: Diagrama de solución propuesta de cómo se integra el sistema de Nagios con Prometheus y un módulo a desarrollar.

Sobre los perfiles de usuarios que consultarán el módulo se evidencian 4 tipos:

- Sysadmin: Este tipo de usuario conoce los sistemas, tiene acceso físico al hardware. En la solución diseñada podrá ver si hay algún problema a nivel de Infraestructura y el estado de los tests de cualquier máquina. Adicionalmente puede crear, eliminar y modificar tests para las máquinas.
- Desarrollador: Este tipo de usuario conoce los servicios provistos por los sistemas, pero no necesariamente le interesa o tiene acceso a los servidores físicamente. Este usuario podrá visualizar el estado de los servicios en cada una de las instituciones.
- Equipo no técnico: Para este tipo de usuario, que tiene la necesidad de saber información sintetizada acerca del estado de todas las instituciones, no necesariamente conoce de los servicios que engloba cada una de ellas. Este usuario podrá acceder al estado general de cada una de las Instituciones, es decir, si están funcionando correctamente o no.
- Usuario público: Para este tipo de usuario, que tiene el menor nivel de privilegios, sólo le interesa saber el nivel de disponibilidad de los servicios. Este usuario podrá ver básicamente si los servicios web están o no disponibles en la institución actual.

Debido a la anterior descripción, se propone la construcción de un modelo de datos y paso a paso en el módulo que contenga los siguientes pasos:

- Instituciones: Son las entidades base, que contienen a los servicios. Representan a cada uno de los clientes del Centro Ucampus.
- Servicios: Son entidades que contienen a las máquinas de la infraestructura y sus tests. Representan a cada uno de los servicios de software necesarios para mantener el software de las plataformas funcionando.

- Infraestructura y Tests: Son entidades que representan a las máquinas y los tests que se realizan sobre ellas. Al tener el estado de un test, este se propaga hasta las instituciones según su estado, es decir, si un test en una infraestructura está fallando, esto afectará a los servicios que le contengan y esto a su vez afectará a las instituciones que contengan dichos servicios.

Para poder responder a este modelo es necesario que la información obtenida desde Nagios a través de Prometheus sea categorizada. La información brindada por Nagios consiste en el nombre del test, el nombre de la máquina sobre la cual está siendo realizado y el resultado del test, es decir un estado que puede ser “Ok”, “Warning”, “Critical” o “Unknown”. Por lo tanto, los datos faltantes en este esquema serían la institución y servicio afectados por cada test.

Finalmente, se propone construir una base de datos inicial, con una única tabla con la siguiente estructura:

TESTS	
TES_HOSTNAME	varchar
TES_DESCRIPCION	varchar
TES_INSTITUCION	varchar
TES_SERVICIO	varchar

Figura 4.3: Estructura de base de datos para la categorización en servicio e institución a los distintos tests de Nagios

Capítulo 5

Implementación

En este capítulo se discutirá la implementación de la solución propuesta, la cual estará dividida en tres secciones: Una relativa a cómo se implementa Nagios y Prometheus; otra respecto a la integración de métricas de Prometheus en el ecosistema de Ucampus y luego, la construcción del módulo de Ucampus como tal. Finalmente, se hablará respecto a Grafana y cómo se implementaron dashboards para visualizar métricas históricas en caso de requerir mayor detalle de lo mostrado en el módulo.

5.1. Comunicación entre Nagios y Prometheus

Para la primera parte de la implementación fue necesaria una manera de integrar la solución ya existente, es decir, el software Nagios a las métricas de Prometheus. Esta integración permitió acceder de forma programática a los eventos actuales y pasados de los tests de Nagios. Para esto se utilizó un software llamado Iapetos [5], cuya labor es integrarse a Nagios y proveer de métricas legibles por Prometheus.

Iapetos es un exporter de Prometheus, esto es, un software que recopila datos de otros sistemas y los disponibiliza en un formato consumible por Prometheus. En este caso, Iapetos es exporter de datos de Nagios, lo cual permite acceder a los tests realizados y sus estados actuales.

Para realizar su instalación, el software se compila o se descarga como binario, se almacena en una ruta accesible por el software Nagios y se le entrega como parámetro de configuración en la siguiente forma:

```
broker_module=/path/to/your/bin/iapetos_naemon config_file=/path/to/  
your/config/config.yaml
```

Como se aprecia en la figura 5.1, al configurar Iapetos, este levanta, por defecto, un servidor HTTP en el puerto 9245. Luego, se modifica el archivo de configuración de Prometheus para que consulte las métricas expuestas por Iapetos en ese puerto.

Con esta configuración se logra que el estado de los tests definidos en Nagios sean visibles

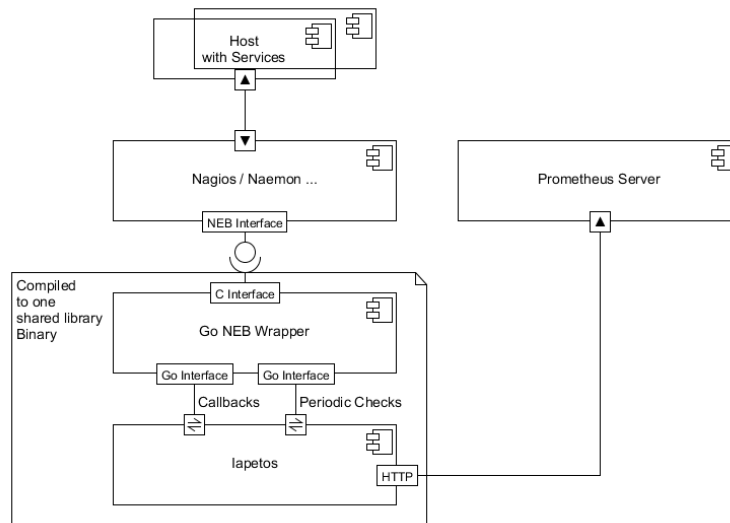


Figura 5.1: Estructura de funcionamiento de Iapetos

en Prometheus y se almacenen sus valores históricos en la base de datos correspondiente. Esto permite acceder a dichos datos de manera programática desde otros softwares utilizando la API provista por Prometheus.

5.2. Integración de Prometheus a Ucampus

Como ya se señaló en el Capítulo 2, el software Prometheus tiene APIs que permiten consultar sobre las métricas almacenadas en su base de datos. De las APIs disponibles se utilizaron dos endpoints:

1. Instant Queries: `/api/v1/query`, que permite consultar el valor de una o varias métricas en un instante de tiempo particular.
2. Range Queries: `/api/v1/query_range`, que permite consultar el valor de una o varias métricas en un rango de tiempo entregado como parámetros.

En ambos casos, las APIs se pueden consultar por solicitudes POST o GET y retornan un JSON. En la implementación se utilizaron siempre solicitudes POST.

Por ejemplo, supongamos que un administrador quiere obtener el número de clicks ocurridos en el último minuto en la institución “ucursos”, la solicitud POST correspondiente a esta consulta al endpoint de Instant Queries y tendría el siguiente contenido:

```
sum(increase(ucampus_clicks_total{institucion="ucursos"}[1m]))
```

El resultado de esta consulta será un JSON, como se muestra en la figura 5.2. La respuesta contiene: el estado de la consulta, el tipo de resultado (en este caso un vector) y un par ordenado con el timestamp y el resultado (en este caso, alrededor de 462 clicks).

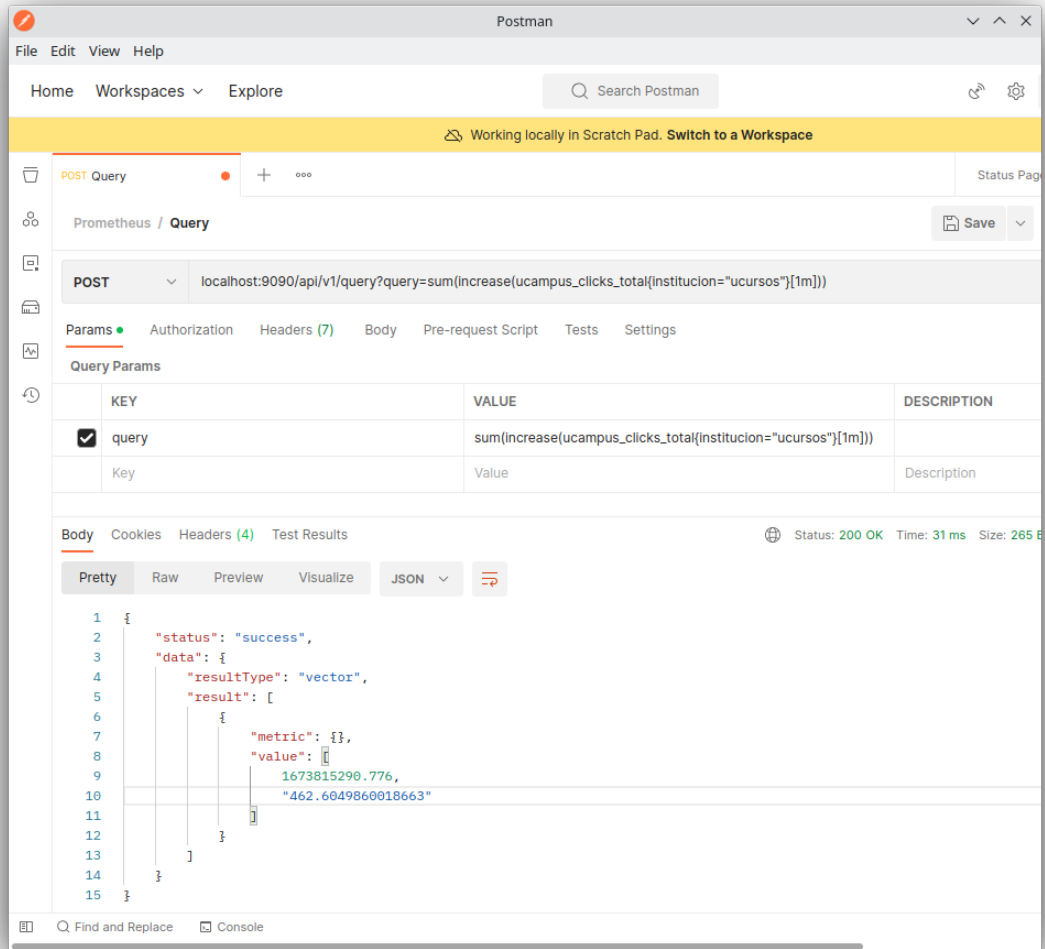


Figura 5.2: Ejemplo de respuesta a una consulta del número de clicks durante el último minuto en la institución “ucursos” en Prometheus

Dada la naturaleza de la información que se quiere mostrar, es importante tener los últimos datos almacenados por Prometheus desde Nagios. Por lo tanto, se creó una librería que encapsula estas peticiones POST para hacer dichas consultas programáticamente.

Se crearon funciones `__query`, `instant_query` y `range_query` que encapsulan las peticiones POST. Además se crearon otras funciones `get_<metrica>` y `get_range_<metrica>` en las que se encapsula la lógica necesaria para consultar una métrica específica.

- `__query($data, $endpoint)` es la función que encapsula una llamada POST genérica dados un `$endpoint` como string y `$data` como un diccionario de parámetros para la llamada POST. Retorna un JSON entregado por la API, como un diccionario de PHP.
- `instant_query($query)` es la función que encapsula las llamadas al endpoint de Instant Queries, recibe `$query` como string, que representa la consulta a realizar a prometheus, en su propia sintaxis de consultas denominada PromQL. Esta función

retorna un diccionario de PHP, donde las llaves son los nombres de cada institución y los valores son los resultados de las consultas.

- `range_query($query, $raw = false, $start_unixtime = 0, $end_unixtime = 0, $step_secs = 60)` es la función que encapsula llamadas al endpoint de Range Queries, recibe `$query` como string, que representa la consulta a realizar en PromQL, `$raw` como booleano para determinar si se pre-procesa o no el JSON retornado, `$start_unixtime` y `$end_unixtime` como enteros, que por defecto retornan la última hora de actividad de la métrica y `$step_secs`, que determina el espaciamiento entre métricas a recibir, que por defecto es cada 60 segundos.

Dadas estas tres funciones genéricas se implementaron las demás funciones de tipo `get_<metrica>` y `get_range_<metrica>` con la lógica específica de consulta de cada una de las métricas.

Usando estas funciones se puede consultar los datos de Prometheus programáticamente desde el sistema Ucampus. Estando disponibles estos datos se procedió a crear un módulo de Ucampus para visualizar los datos de las consultas.

5.3. Módulo de Ucampus

Se implementó un nuevo módulo de Ucampus para visualizar los valores de tests y de distintas métricas almacenadas en Prometheus donde un usuario es capaz de configurar tests y relacionarlos con servicios e instituciones a las que sirven; también, el usuario puede visualizar el estado de las instituciones y servicios asociados a cada institución, según sea el estado de los tests que contemplen.

Como se dijo en el Capítulo 2.4, los módulos de Ucampus siguen un paradigma MVC. En esta sección se revisará cómo se implementó la lógica del módulo y sus funciones más importantes.

El módulo cuenta con una Base de datos que almacena los nombres de los tests, los host-names de dichos tests, el servicio al que corresponden y las instituciones a las que pertenece. Se implementaron las siguientes funciones para interactuar con la base de datos:

- `get($institucion = '', $servicio = '')` que recibe opcionalmente `$institucion` y `$servicio` como strings para filtrar los tests obtenidos desde la base de datos según institución y/o servicio. Esta función retorna un diccionario de diccionarios, cuya llave es un identificador alfanumérico y su contenido es un diccionario con las columnas de la tabla y su respectivo valor.
- `norm($t)` que recibe un diccionario de PHP representando una fila de la base de datos y normaliza el contenido de dichas filas. Retorna un diccionario de PHP con los datos iniciales normalizados, más otros campos generados a partir de ellos.
- `upd($t)` que recibe un diccionario de PHP representando un test en particular y lo busca según su llave en la base de datos y actualiza los campos que presenten cambios según sea necesario. Esta función no retorna ningún valor.

- `del($t)` que recibe un diccionario de PHP representando un test en particular y lo elimina de la base de datos. Esta función no retorna ningún valor.

Además se implementó la función `getTests` que consulta tanto a la base de datos como a Prometheus sobre los tests de Nagios y sus estados y consolida dicha información en una única estructura en un diccionario de PHP. Esto permite que al consultar esta función se sepa sobre tests nuevos que aún no han sido configurados y también sobre tests configurados previamente en la base de datos que ya no aparecen existentes en Prometheus.

Listing 5.1: Código de la función `getTests`

```

1 <?php
2
3 function getTests( $institucion = '', $servicio = '', $hostname = '' )
4     {
5         $tests_p = PrometheusAPI::get_tests();
6         $tests_db = Tests::get( $institucion );
7
8         $tests = [];
9         foreach( $tests_db as $t ) {
10            if( $servicio && $t['servicio'] != $servicio ) continue;
11            if( $hostname && $t['hostname'] != $hostname ) continue;
12
13            foreach( $t['instituciones'] as $i ) {
14                $tests['by_institucion'][$i][ $t['servicio'] ][] = $t[
15                    'id'];
16                $tests['by_hostname'][$t['hostname']][$i][] = $t['id
17                    '];
18                $tests['by_servicio'][$t['servicio']][$i][] = $t['id
19                    '];
20            }
21            if( ! $tests['by_servicio'][$t['servicio']] ) continue;
22
23            $tests['by_id'][$t['id']] = $t + [
24                'value' => isset( $tests_p[ $t['id'] ]['value'] ) ?
25                    $tests_p[ $t['id'] ]['value'] : -1

```

Al acceder al módulo por primera vez, no se encontrarán instituciones ni servicios configurados. Para configurarlos, hay que ingresar a la página de configuración, en donde se listarán los tests retornados por Prometheus y se les podrá asignar un nombre de servicio al que pertenezcan y los nombres de instituciones a los que sirvan.

Luego de configurados los tests, en la vista de Instituciones se mostrará el estado de cada una de ellas según sea el estado de los servicios que contenga y en la vista de Servicio, el detalle de cuáles tests contiene y el estado de cada uno.

A continuación revisaremos las vistas del módulo, qué información muestra cada una y cómo se navegan.

La vista de instituciones, mostrada en la figura 5.3 es la vista inicial y más general de todas. Muestra métricas acumuladas de los servicios que posee cada una, el estado general de los tests y algunas métricas importantes como los clicks y los conectados.

Monitoreo

Instituciones Configuración

Instituciones 12

N°	Institución	Estado	Conectados	Clicks por min	Tests	Servicios
1	U-Cursos UChile	OK	5.087	1.689	54/54	5/5
2	UChile	OK	1.268	501	5/5	3/3
3	UOH	OK	815	327	19/19	5/5
4	UMCE	OK	174	48	20/20	5/5
5	UAysén	OK	92	20	19/19	5/5
6	UARcoleta	OK	7	5	22/22	5/5
7	INAF	OK	29	7	20/20	5/5
8	Gendarmería	OK	17	0	22/22	6/6
9	Doñihue	OK	1	0	24/24	6/6
10	Súbete	OK	6	7	22/22	5/5
11	Ucampus.cl	OK	-	-	3/3	2/2
12	Infra Ucampus	FAIL	-	-	3/6	1/4

Figura 5.3: Vista inicial del módulo desarrollado, mostrando el estado general de las instituciones

Al hacer click sobre el nombre de una de las instituciones, se muestra la vista general de servicios de esa institución, mostrada en la figura 5.4, en forma de matriz, que permite identificar tanto los servicios asociados a la institución como las máquinas que los sirven. Abajo de la matriz se puede observar las métricas de Conectados y Clicks manuales realizados por los usuarios de dicha institución en la plataforma. En esta vista es posible clicar cada uno de los servicios, las máquinas o el número de tests, en cuyo caso se redirige a otra vista del módulo.

En el caso de clicar el número de tests en la matriz, se carga una nueva vista con el detalle de los tests que componen a dicho servicio en la institución. Esto se puede ver en la figura 5.5.

En el caso de clicar un servicio de la vista de instituciones, se carga una nueva vista, que muestra ese servicio en todas las instituciones que le contengan y los tests que componen cada servicio. Esto se puede ver en la figura 5.6.

En el caso de clicar uno de los hostnames, se mostrará la vista de hostname, como se ve en la figura 5.7. Es posible observar en esta vista todos los tests realizados sobre esta máquina, el servicio al que pertenecen y las instituciones a las que sirve. Además, al costado derecho es visible el estado de cada uno de los tests.

Finalmente está la vista de configuración, visible en la figura 5.8 que permite asignarle nombre de servicio e instituciones a cada test que Prometheus retorna que aún no ha sido configurado. Para los tests ya almacenados, esta vista permite eliminarlos de la base de datos, de forma que puedan ser reagregados en un futuro con los datos correctos.

Monitoreo

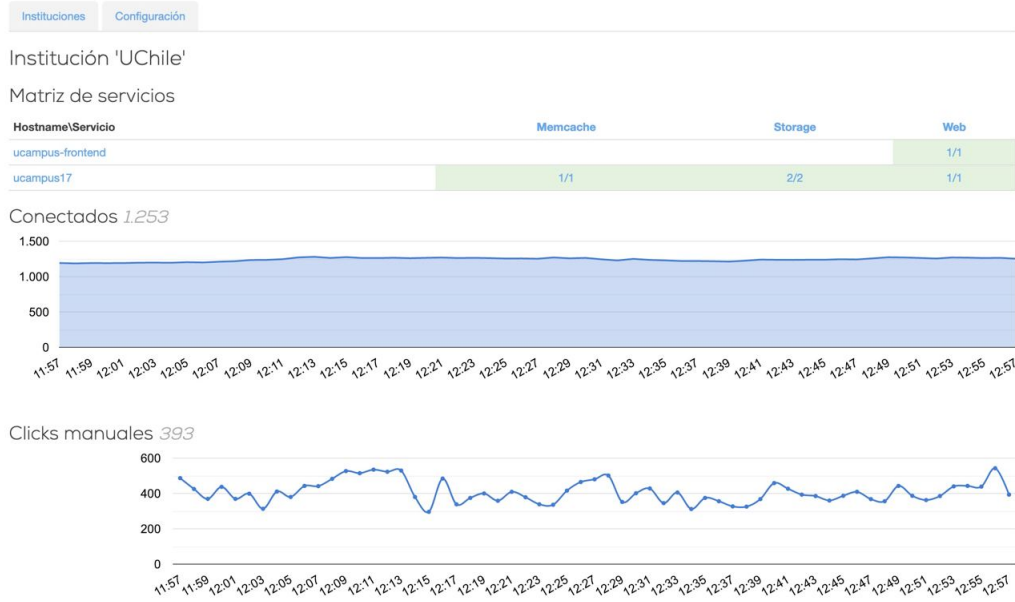


Figura 5.4: Vista detallada de una institución en el módulo desarrollado

Monitoreo

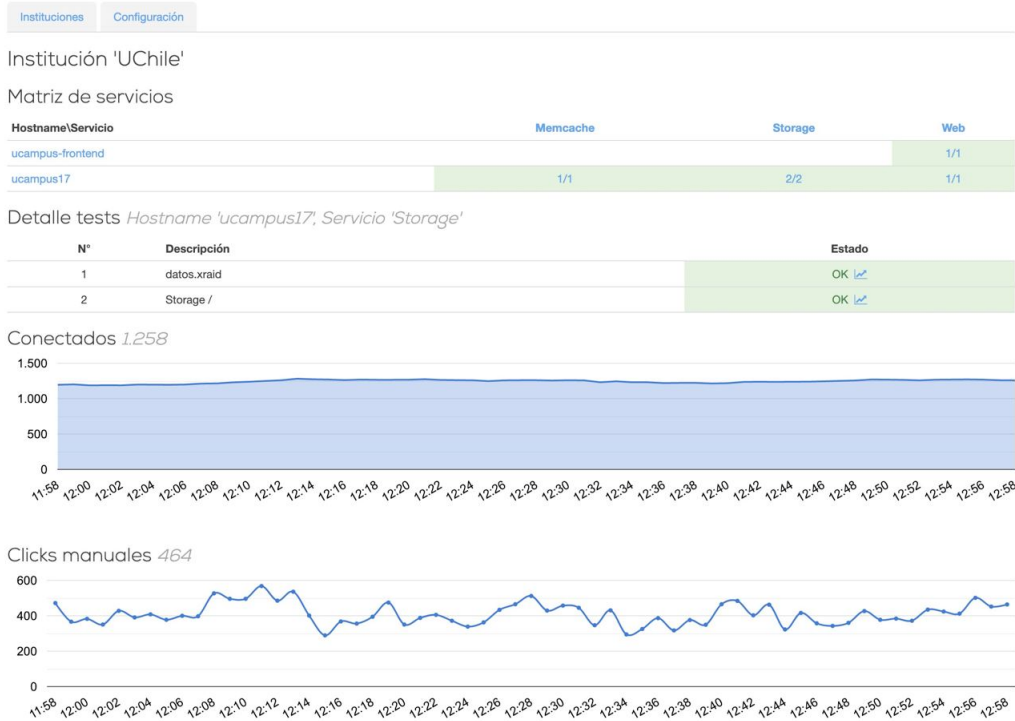


Figura 5.5: Vista de una institución con detalle de un servicio en particular

En las vistas de las figuras 5.5, 5.6 y 5.7 es posible observar que al lado del estado 'OK' de cada test hay un ícono de gráfico pequeño. Dicho ícono es un enlace a un dashboard de visualización del software Grafana que muestra el estado actual e histórico de cada test

Monitoreo

Instituciones Configuración

Servicio 'Memcache'

N°	Hostname	Descripción	Estado
Doñihue			
1	donihue01	Memcache Checker	OK ↗
2	donihue02	Memcache Checker	OK ↗
Gendarmería			
3	gendarmeria01	Memcache Checker	OK ↗
4	gendarmeria02	Memcache Checker	OK ↗
INAF			
5	inaf01	Memcache Checker	OK ↗
6	inaf02	Memcache Checker	OK ↗
Súbete			
7	subete01	Memcache Checker	OK ↗
8	subete02	Memcache Checker	OK ↗
UAR Recoleta			
9	uar01	Memcache Checker	OK ↗
10	uar02	Memcache Checker	OK ↗
UAysén			
11	uaysen01	Memcache Checker	OK ↗
U-Cursos UChile			
12	ucursos31	Memcache Checker	OK ↗
13	ucursos32	Memcache Checker	OK ↗
14	ucursos41	Memcache Checker	OK ↗
15	ucursos42	Memcache Checker	OK ↗
UMCE			
16	umce01	Memcache Checker	OK ↗
17	umce02	Memcache Checker	OK ↗
UOH			
18	uoh01	Memcache Checker	OK ↗
UChile			
19	ucampus17	Memcache Checker	OK ↗

Figura 5.6: Vista de un servicio en todas las instituciones

Monitoreo

Instituciones Configuración

Host 'ucampus17'

N°	Servicio	Descripción	Estado
UChile			
1	Web	HTTP	OK ↗
2	Memcache	Memcache Checker	OK ↗
3	Storage	datos.xraid	OK ↗
4	Storage	Storage /	OK ↗

Figura 5.7: Vista de una máquina o hostname en particular

haciendo consultas a la misma instancia de Prometheus utilizada por el módulo. En la figura 5.9 se puede observar el dashboard de Grafana al cual lleva cada test.

Además, se crearon dashboards de Grafana para la visualización de las métricas de clicks

Monitoreo

Instituciones Configuración

Tests no Configurados

Guardar

Nº	Hostname	Descripción	Institución	Servicio
1	SW10G01	host_check	Select Some Option:	<input type="text"/>
2	SW10G02	host_check	Select Some Option:	<input type="text"/>
3	ar54	host_check	Select Some Option:	<input type="text"/>
4	ar54	LOAD	Select Some Option:	<input type="text"/>
5	ar54	Ping	Select Some Option:	<input type="text"/>
6	dev	host_check	Select Some Option:	<input type="text"/>
7	dev	Hora	Select Some Option:	<input type="text"/>
8	dev	LOAD	Select Some Option:	<input type="text"/>
9	dev	Ping	Select Some Option:	<input type="text"/>
10	dev	DNS Resolve	Select Some Option:	<input type="text"/>

Figura 5.8: Vista de configuración

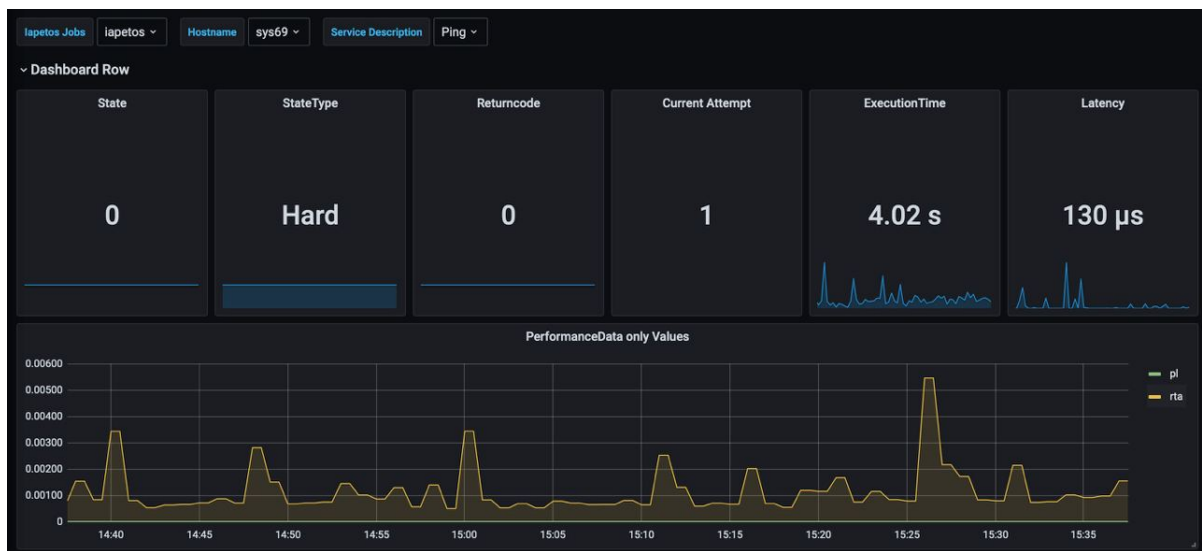


Figura 5.9: Dashboard de Grafana para tests de Nagios

y conectados de cada institución, que se pueden observar en las figuras 5.10 y 5.11

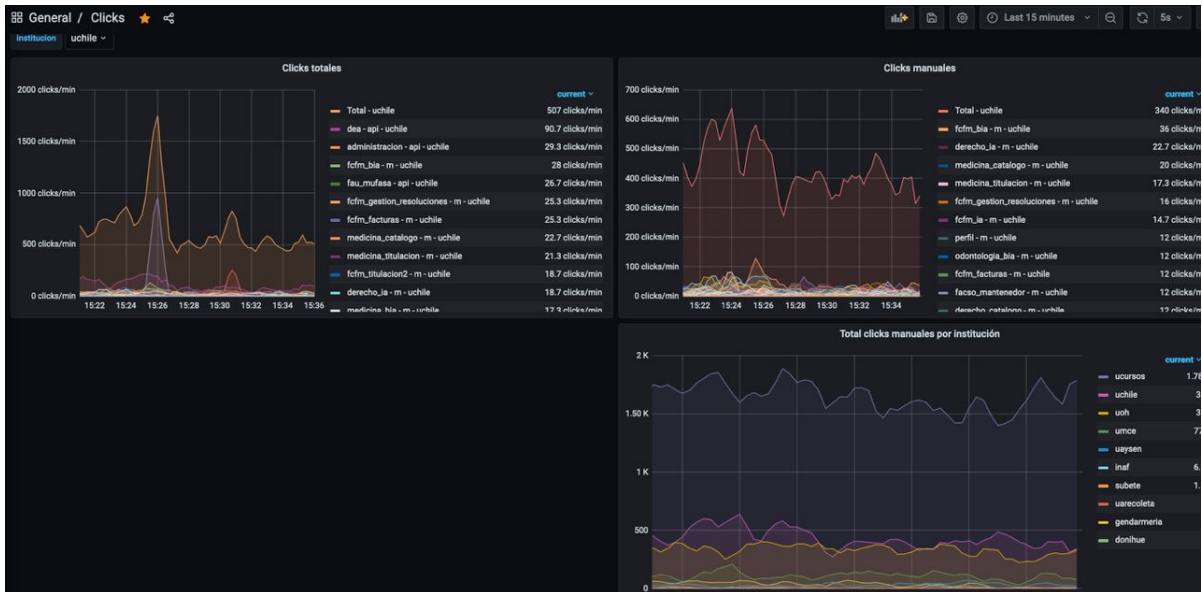


Figura 5.10: Dashboard de Grafana para visualizar clicks de usuarios en cada institución

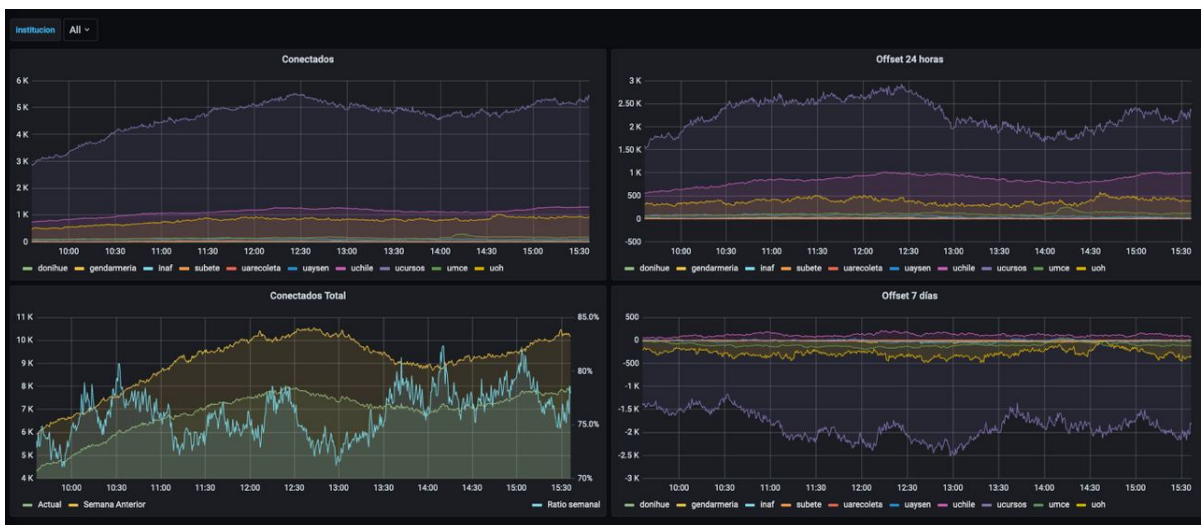


Figura 5.11: Dashboard de Grafana para visualizar la cantidad de usuarios conectados en cada institución

5.4. Validación

Durante el proceso de validación, se evaluarán las mejoras de funcionalidad y usabilidad que provee el trabajo realizado.

5.4.1. Revisión de Problemas

A lo largo de esta memoria hemos evidenciado y descrito distintos problemas que el equipo Ucampus presentaba en cuanto a monitoreo y manejo de incidencias. A continuación, vamos a listar cada uno de estos problemas y describirlos nuevamente, para constatar como se afrontaban anteriormente y contrastarlo con como está siendo realizado con la herramienta desarrollada en esta memoria.

Uno de los problemas más evidentes a lo largo de la memoria ha sido la falta de registros históricos de los sistemas. Por ejemplo, no se sabía cual era el estado normal de carga de la plataforma. Es decir, si en un periodo de inscripción académica (IA) se sobrecargan los sistemas, no era posible saber si se encontraba en niveles anómalos o no. Otra consecuencia de esta situación es que el análisis posterior de cada evento (post-mortem) se vuelve más complejo de analizar.

Debido a las experiencias anteriores con el mismo ejemplo de la IA, era conocido que ante una sobrecarga se iba en busca de la información a los distintos scripts de la interfaz de herramientas para sysadmin de la plataforma, mostrada en la figura 5.12. Dentro de esa lista de elementos visitados, usualmente el de mayor utilidad era la vista en vivo de los clicks, mostrada en la figura 5.13.

Con la herramienta desarrollada se agrega Prometheus que cumple la función de almacenar un registro histórico, y por lo tanto, para el mismo problema de una inscripción académica se ha hecho posible comparar el estado actual con el pasado y el comportamiento de distintas métricas para un mismo periodo de tiempo.

Como se mencionó en un párrafo anterior, ante alguna incidencia se realizaba una búsqueda de información disponible, es decir, se visitaban distintos scripts para lograr hacerse una idea del problema. Por lo tanto, otro de los problemas era de que se poseían herramientas desacopladas.

Si bien la herramienta desarrollada no reemplaza hoy en día todos los lugares con información anteriormente mencionados, presenta un lugar extensible en donde centralizar dicha información. Ejemplos de estas integraciones implementadas actualmente son los clicks, los conectados y la integración con dashboards de Grafana. Dichas integraciones se muestran en las figuras 5.10 y 5.11 de la sección de implementación.

Otro de los problemas presentes en Ucampus es que no es directa la relación entre los servidores y las instituciones a las que sirve cada uno. Además de la complejidad añadida por los servidores virtualizadores, que contienen múltiples servidores virtuales, sirviendo potencialmente a varias instituciones. Entonces, puede no ser inmediata la identificación de las instituciones afectadas por una falla en un servidor.

Administración

Grupos Módulos Personas Parámetros Scripts Logs SYS

Sysadmin Stuff

- [Servidor Actual: ucampus81-int](#)
Servidor Actual: **ucampus81-int**
- [Check de Instalación Global](#)
Acá se revisan las variables específicas del sistema para saber que está todo ok instalado.
- [Check de Instalación Local](#)
Acá se revisan las variables específicas del sistema para saber que está todo ok instalado.
- [Live URLs](#)
Visión el tiempo real de las Urls que llaman los usuarios.
- [Changelog](#)
Logs de los commit en SVN.
- [OPCache](#)
Aca se puede controlar el estado del Cache de PHP.
- [Balanceador](#)
Desde aquí se puede controlar los workers.
- [Memcache](#)
Aca se puede ver el estado de Memcached, así como caducar las entradas en caso de que se requiere forzar un recache. También puedes consultar el [script oficial](#).
- [Manticore](#)
Estadísticas del Manticore.
- [Información de PHP](#)
Información sobre la instalación de PHP en el servidor.
- [Requests de Apache](#)
Lista de conexiones a los servidores.
- [Monitores](#)
Lista de monitores de instalación y carga.
- [Nagios](#)
Nagios.
- [Clicks de Ucampus](#)
Mapa de clicks en tiempo real por módulo (sin llamadas de apis).
- [Clicks de U-Cursos](#)
Los clicks.

Figura 5.12: Interfaz web de herramientas para sysadmin de la plataforma Ucampus

En la herramienta desarrollada, la interfaz inicial muestra un estado global de todas las instituciones, mostrando cada institución, su estado general y algunas métricas relevantes. En la figura 5.3 se muestra cómo se ve esta interfaz.

Como ya se ha mencionado anteriormente, en el equipo de Ucampus existen distintas áreas y dentro del área de desarrollo existen dos roles principales: Sysadmin, encargado de administrar los servidores y mantener sus sistemas operativos y softwares al día; y Desarrollador, encargados de mantener, actualizar y desarrollar nuevo software para las plataformas. Dado que no toda la información relevante para un sysadmin lo es para un desarrollador, los correos de notificación sobre errores en la plataforma son enviados a los sysadmin.

Considerando lo anterior, falta información sintetizada acerca del estado de los sistemas. Esta información serviría a los desarrolladores para entender en términos más simples el estado de los servicios y para un sysadmin para localizar de manera más rápida los sistemas afectados por una falla.

La herramienta desarrollada resuelve esta necesidad por medio de un conjunto de vistas que muestran con cada vez mayor detalle el estado de los servicios y sistemas subyacentes a cada institución. Estas vistas proveen entonces de información sintetizada y rápida para un sysadmin y mayores detalles acerca del estado de los servicios a un desarrollador que las herramientas existentes.

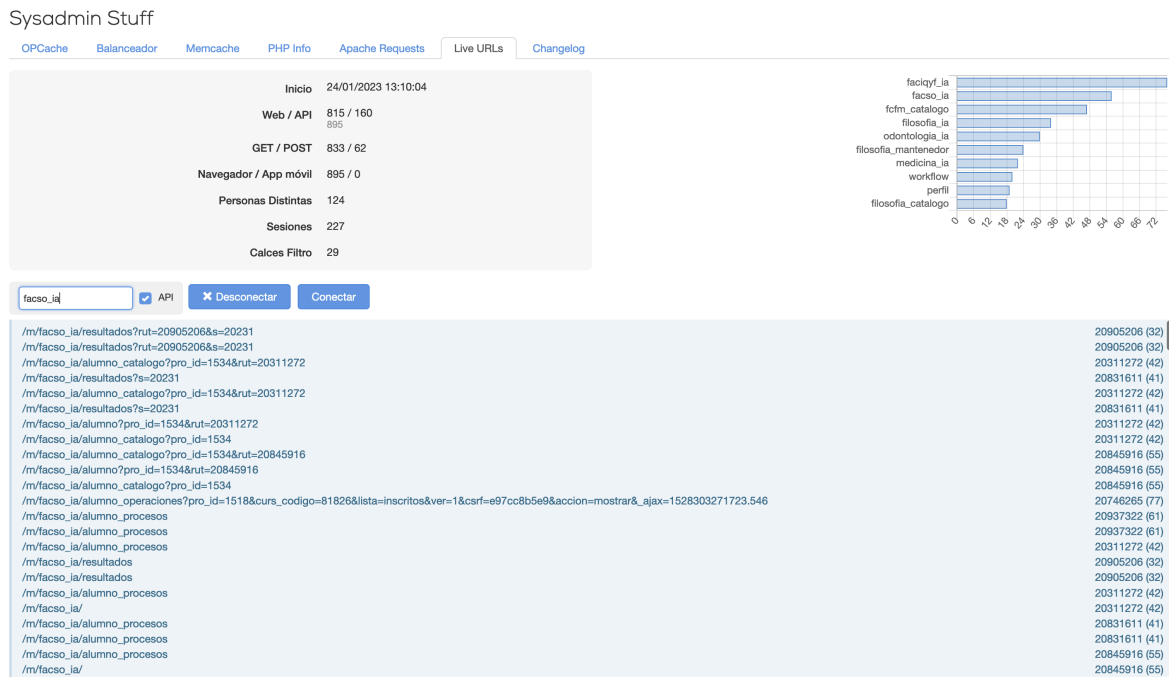


Figura 5.13: Interfaz web de vista en vivo de los clicks realizados en la plataforma Ucampus

5.4.2. Validación con Equipo

Durante el transcurso del trabajo de memoria se realizaron dos instancias de validación con el equipo de desarrollo del Centro Ucampus. En la primera de estas instancias, en semana 4, se mostraron mockups de un diseño inicial del módulo, con visualizaciones de muestra y datos de demostración. En esta reunión el equipo sugiere que en las vistas de institución sea posible tener una matriz de servicios que muestre los servicios y las máquinas que proveen cada uno de ellos a esa institución. Posteriormente, durante el desarrollo de la implementación funcional se agregó esta característica.

Durante la segunda instancia de validación, en semana 10, se muestra el módulo funcional al equipo de desarrollo del Centro Ucampus, quienes entregan feedback respecto a la conectividad entre vistas, en las que no era claro cómo llegar de una vista a otra que fuera interesante debido a la falta de enlaces internos entre ellas. Además se realiza la observación de que la vista de configuración podría ser simplificada, con el fin de mejorar la velocidad de carga y también para hacerla menos engorrosa al agregar nuevos tests. Posteriormente se realizaron cambios en las vistas, creando enlaces en las vistas de servicios, institución y hostname entre ellas, de manera que sea fácil llegar de una a las otras en caso que fuese necesario investigar la ubicación de un problema, por ejemplo.

También en la segunda instancia se hizo notar que la idea de contar con una vista pública del módulo, con información simplificada sobre el estado interno de los servicios, sería útil no solo para los clientes de la plataforma, sino también para usuarios. Esta idea se superpone en varios ámbitos a la idea de una página de status pública, y mediante el sistema de permisos que provee Ucampus, se puede segmentar la visualización del módulo, de forma que sólo sea visible parte de la información, dependiendo de los privilegios que se tengan dentro

del sistema. Con esto se lograría un módulo que provea información detallada al equipo de desarrollo y también información simplificada a los usuarios de la plataforma o a externos que no tengan credenciales en los sistemas.

Capítulo 6

Conclusión

En esta memoria se ha trabajado en comprender el estado actual del monitoreo de infraestructura y servicios del Centro Ucampus. Se han estudiado soluciones y acercamientos tanto del “mercado” como personas con experiencia en otras empresas de desarrollo de software. Se han analizado y dividido los problemas en dos secciones, Monitoreo e Incidencias, y se ha reconocido una gran oportunidad de mejora en la primera de estas por lo que se decidió abordarla descartando mejoras considerables en lo relativo a incidencias.

Para abordar el problema se construyó, a partir de herramientas tanto del estado del arte como de un desarrollo propio, un proceso y sistema idóneo para solucionar parte de los problemas encontrados y capaz de ser extendido para poder permitir mejoras futuras. Este sistema fue validado con el equipo de desarrollo de Ucampus dándose a conocer posibles mejoras al sistema y una manera distinta de abordar la configuración.

Debido a lo anterior, se concluye que tanto el objetivo general como los objetivos específicos se cumplieron satisfaciendo a los usuarios y clientes. Aun así, parte de la retroalimentación en la última etapa de validación y la necesidad de un sitio público no alcanzó a ser entregado ni validado para abordar completamente lo considerado en las etapas de análisis.

Además, se concluye que al trabajar con un equipo de desarrollo de software permitió que la validación de este fuese valiosa, precisa y mucho más directa comparativamente con lo que se podría obtener de algún equipo de distinta índole.

Asimismo, el hecho de que Ucampus haya creado su propio framework de desarrollo permitió implementar una herramienta tanto de front-end como back-end con mayor claridad para enfocarse en el problema específico destinado por la memoria. Si bien esto en parte se debe al paradigma MVC, también es relevante la separación modular que poseen las plataformas de Ucampus.

Una de las conclusiones personales de este proyecto de memoria fue la necesidad de comunicar de mejor manera y con claridad las ideas y, por ende, la falta de tiempo planificado para la escritura de la memoria. Se observa que debe ser una habilidad a mejorar para poder desenvolverse en proyectos futuros.

Por otra parte, personalmente la visión que se poseía acerca del hardware necesario para

la mantención de una infraestructura a la escala de Ucampus cambió luego de comprender la capacidad real de los sistemas dedicados desarrollados y la optimización de código. Usualmente se poseía la perspectiva de otros desarrollos en la nube o también el sesgo de un stack con un lenguaje interpretado como PHP, en donde se esperaba lentitud y una dimensión de hardware mayor. Esto viene a confirmar que es más eficiente en este caso el desarrollo de “productos” propios que tercerizar las necesidades, sobre todo considerando la cantidad de herramientas que se disponían en el estado del arte.

6.1. Trabajo Futuro

Como se mencionó en la sección 5.4, parte de la última retroalimentación por parte del equipo Ucampus fue la incomodidad al poseer múltiples *tests* que deben ser categorizados en primera instancia. Por ende, una mejora futura directa vendría a ser la necesidad de implementación o refactorización del modelo para categorizar. Aun así, un primer acercamiento fue realizado insatisfactoriamente debido al surgimiento de casos de borde cuando *tests* son eliminados del origen de Nagios o nuevos son agregados. Es decir, este trabajo futuro debe tener presente que quizás una generalización, aunque suene contradictorio, puede conllevar a eliminar flexibilidad en otros aspectos.

Por otra parte, el sistema actualmente reporta gravedad al fallar un test pero en ocasiones existen algunos test que podrían presentar un nivel de alerta distinta como por ejemplo solo la necesidad de ser observado o que solo los desarrolladores deban realizar alguna acción, y no así una alerta para todos los usuarios. Dado lo anterior puede extenderse el modelo para categorizar bajo distintos niveles de gravedad entre distintos usuarios.

Finalmente, se evidencia que en las últimas ocasiones en que se han producido incidencias, si bien la alerta se observa en el módulo desarrollado y lo histórico puede ser analizado en Grafana, sería útil un resumen gráfico de una temporalidad corta dentro del módulo para poseer una vista rápida y directa al producirse un evento anómalo.

Bibliografía

- [1] Cachet - an open source status page system for everyone. <https://github.com/CachetHQ/Cachet>. Visitado el 18-12-2022.
- [2] Centro tecnológico ucampus. <https://ucampus.cl/>. Visitado el 10-08-2022.
- [3] Elk - el elk stack: de los creadores de elasticsearch. <https://www.elastic.co/es/what-is/elk-stack>. Visitado el 18-12-2022.
- [4] Grafana: The open observability platform. <https://grafana.com/>. Visitado el 18-12-2022.
- [5] Iapetos - a nagios / naemon prometheus exporter. <https://github.com/Griesbacher/Iapetos>. Visitado el 18-12-2022.
- [6] Iso 9001 - wikipedia. https://es.wikipedia.org/wiki/ISO_9001. Visitado el 10-08-2022.
- [7] Jira - software de seguimiento de proyectos e incidencias. <https://www.atlassian.com/es/software/jira>. Visitado el 10-08-2022.
- [8] Lamp - wikipedia. <https://es.wikipedia.org/wiki/LAMP>. Visitado el 10-08-2022.
- [9] Monitoring - wikipedia. <https://en.wikipedia.org/wiki/Monitoring#Computing>. Visitado el 10-08-2022.
- [10] Nagios - the industry standard in it infrastructure monitoring. <https://www.nagios.org/>. Visitado el 10-08-2022.
- [11] Netdata - real-time performance monitoring, done right! <https://github.com/netdata/netdata>. Visitado el 18-12-2022.
- [12] Odoo - the #1 open source crm software. https://www.odoo.com/es_ES/app/crm. Visitado el 18-12-2022.
- [13] Prometheus - monitoring system and time series database. <https://prometheus.io/>. Visitado el 18-12-2022.
- [14] Salesforce - ¿qué es un crm? <https://www.salesforce.com/mx/crm/>. Visitado el 18-12-2022.
- [15] Slack es tu sede digital. <https://slack.com/intl/es-cl/>. Visitado el 18-12-2022.

- [16] Statping - status page for monitoring your websites and applications with beautiful graphs, analytics, and plugins. run on any type of environment. <https://github.com/statping/statping>. Visitado el 18-12-2022.
- [17] Telegram messenger. <https://telegram.org/>. Visitado el 10-08-2022.
- [18] Uptime kuma - a fancy self-hosted monitoring tool. <https://github.com/louislam/uptime-kuma/>. Visitado el 18-12-2022.
- [19] Whatsapp. <https://www.whatsapp.com/>. Visitado el 18-12-2022.