

SSL をはじめよう

「なんとなく」から「ちゃんとわかる！」へ

mochikoAsTech 著

2020-03-01 版 **mochikoAsTech 発行**

はじめに

2020 年 3 月 mochikoAsTech

本著を手に取ってくださったあなた、こんにちは！ あるいは、はじめて。『SSL をはじめよう』の筆者、mochikoAsTech です。

SSL は好きですか？ それとも怖いですか？ 本著を書くまで、筆者は「ちゃんと分かっているとは言えないので、SSL を迂闊にさわるのはなんだか怖い」と感じていました。

SSL は、エンジニアのごく身近なところにあります。たとえば「サイトをフル SSL 化する」「SSL 証明書を更新する」のような形で、誰しも一度は SSL と関わりをもったことがあるのではないでしょうか。しかし関わる機会が多い割に、「ちゃんと分かっているか」と言われるとちょっと自信がない、そんなエンジニアは、筆者を含め、きっと一定数いること思います。

Google が 2014 年ごろからさかんに提唱している「HTTPS everywhere」（直訳すると「どこでも HTTPS」）のもと、ウェブサイトの評価基準は「HTTP で 100 点、HTTPS にするとさらに加点」から、「HTTPS で 100 点、できていなければ減点」に変わってきました。ますます SSL と関わる機会が増えてきて、心のどこかで「SSL、ちゃんと分かりたいなあ…」と思っている。本著はそんな人のための一冊です。

理解度は「分かった！」「分かってなかった…」をくり返して、段々と上がっていくものです。まずは本著で、一緒に最初の「分かった！」まで進んでみましょう。

ちなみに本著「SSL をはじめよう」（以下 SSL 本）では、「DNS をはじめよう」（以下 DNS 本）で購入したドメイン名を使用します。DNS 本を読まずに SSL 本を読み進めていくと、第 2 章辺りで「ここで事前にあく抜きしておいた箇を取り出します」と言われて、「は？ あく抜きとかいつしてたの？！」という状態になります。「DNS は興味ないし面倒くさいんだけど…」という方も、できれば DNS 本を先にお読みいただいて、箇の下ごしらえ（＝ドメイン名の購入）を済ませた状態で SSL 本を開いてみてください。きっとその方が美味しくお召し上がりいただけます。なお SSL 本の第 1 章は、DNS 本を読んでいなくても問題ない内容ですので、とりあえずそのまま読み進めていただいても構いません。

またインフラやサーバに関する説明は、すでに「AWS をはじめよう」（以下 AWS 本）

で行なっていますので、本著では最低限にとどめています。本著で HTTPS のサイトを作ってみて、「インフラちょっと楽しいかも」と思われた方は、よかつたら後で AWS 本も召し上がってみてください。

DNS 本、AWS 本、そして SSL 本のはじめよう 3 部作は、「サーバやインフラは怖いものではなくすごく楽しいものなんだよ」ということを、かつての私のようなインフラ初心者へ伝えたくて書いたシリーズです。

読んで試して「面白かった！」と思ってもらえたなら、そしてインフラを前より少しでも好きになってもらえたなら何より嬉しいです。

想定する読者層

本著は、こんな人に向けて書かれています。

- よく分からぬまま SSL を使っている人
- 「サイトを HTTPS 化したいな」と思っている人
- 証明書の購入や設置の流れがいまいち分かっていない人
- SSL と TLS の関係性がよく分からぬ人
- SSL 証明書が一体何を証明しているのか知らない人
- これからシステムやプログラミングを学ぼうと思っている新人
- ウェブ系で開発や運用をしているアプリケーションエンジニア
- 「インフラがよく分からぬこと」にコンプレックスのある人

マッチしない読者層

本著は、こんな人が読むと恐らく「not for me だった…（私向けじゃなかった）」となります。

- SSL/TLS の通信を C 言語で実装したい人
- 「プロフェッショナル SSL/TLS」を読んで完全に理解できた人

本著の特徴

本著では実際にサーバを立てて SSL 証明書の設置を行い、HTTPS のサイトを作ってみます。手を動かして試してから仕組みを学べるので理解がしやすく、インフラ初心者でも安心して読み進められる内容です。Oracle Cloud の無料枠の中でサーバを立てて使用

しますので、サーバ代はかかりません。SSL 証明書代のみ、1,100 円（税込）かかります。
また SSL をめぐって実際にやってしまいがちな失敗、トラブルをとり上げて、

- こんな障害が起きたら原因はどう調べたらいいのか？
- 問題をどう解決したらいいのか？
- どうしたら事前に避けられるのか？

を解説するとともに、実際にコマンドを叩いて反復学習するためのドリルもついています。

本著のゴール

本著を読み終わると、あなたはこのような状態になっています。

- SSL 証明書がどんな役割を果たしているのか説明できる
- 証明書を買うときの手順が分かっている
- 意図せず「保護されていない通信」と表示されてしまったときの対処法が分かる
- 障害が起きたときに原因を調査できる
- 読む前より SSL が好きになっている
- SSL/TLS と併記されている「TLS」の意味が分かっている

免責事項

本著に記載されている内容は筆者の所属する組織の公式見解ではありません。

また本著はできるだけ正確を期すように努めましたが、筆者が内容を保証するものではありません。よって本著の記載内容に基づいて読者が行った行為、及び読者が被った損害について筆者は何ら責任を負うものではありません。

不正確あるいは誤認と思われる箇所がありましたら、必要に応じて適宜改訂を行いますので GitHub の Issue や Pull request で筆者までお知らせいただけますと幸いです。

<https://github.com/mochikoAsTech/startSSL>

目次

はじめに	3
想定する読者層	4
マッチしない読者層	4
本著の特徴	4
本著のゴール	5
免責事項	5
第1章 Oracle Cloud のアカウントを作ろう	11
1.1 サーバを立てる下準備	12
1.1.1 サイトを作るのにどうしてサーバがいるの？	12
1.1.2 サーバを立てるにはお金が必要？	13
1.1.3 なんで AWS じゃなくて Oracle のクラウドを使うの？	13
1.2 Oracle Cloud でアカウント登録	14
1.2.1 無料でアカウントを作ろう	14
1.2.2 〈トラブル〉 どうしても SMS が届かない！ そんなときは？	21
1.2.3 パスワードと支払情報の登録	21
1.2.4 Oracle Cloud のコンソールにサインイン	26
第2章 Oracle Cloud でサーバを立てよう	29
2.1 事前準備	30
2.1.1 Windows で RLogin をインストールする	30
2.1.2 Windows で SSH のキーペア（秘密鍵・公開鍵）を作成する	33
【コラム】 SSH の秘密鍵にパスフレーズは設定すべき？	37
2.1.3 Mac でターミナルを準備する	38
2.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する	40
【コラム】 ターミナルでコピー&ペーストするには？	41

2.2	コンピュートでサーバを立てる	42
2.2.1	OS は Oracle Linux 7.7 を使おう	43
2.2.2	作っておいた SSH の公開鍵を設置しよう	44
2.2.3	〈トラブル〉 "Out of host capacity." が起きたらどうすればいい？	44
2.2.4	Always Free ではなく無償クレジットの枠でサーバを立てよう	46
2.2.5	サーバが起動するまで待とう	48
	【コラム】 Oracle Cloud のコンピュートの金額計算方法	49
	【コラム】 Oracle Cloud と AWS はどっちが安い？	50
2.2.6	接続先となるサーバの IP アドレス	50
第 3 章	ウェブサーバの設定をしよう	53
3.1	サーバに SSH でログインしよう	54
3.1.1	Windows の RLogin を使ってサーバに入ってみよう	54
3.1.2	Mac のターミナルを使ってサーバに入ってみよう	64
3.2	ターミナルでサーバを操作・設定してみよう	66
3.2.1	プロンプトとは？	67
3.2.2	コマンドは失敗したときだけエラーを吐く	68
3.2.3	ターミナルを閉じたいとき	69
3.3	NGINX をインストールしよう	69
3.4	Firewalld で HTTP と HTTPS を許可しよう	71
3.5	SELinux を無効にしておこう	72
3.6	OS を再起動してみよう	75
3.6.1	ターミナルはなんのためにある？	76
3.7	なぜかサイトが見られない	77
3.7.1	サーバの手前にあるファイアウォールにも穴を空けよう	78
3.7.2	今度こそ HTTP でサイトを見てみよう	82
3.8	ドメイン名の設定をしよう	83
第 4 章	SSL 証明書を取得しよう	87
4.1	SSL 証明書にまつわる登場人物	88
4.2	秘密鍵 (startssl.key) を作ろう	88
	【コラム】 SSL 証明書の秘密鍵にパスフレーズは設定すべき？	89
4.3	CSR (startssl.csr) を作ろう	90
4.3.1	【ドリル】 CSR で入力すべきなのはクライアントの情報？	92
4.4	SSL 証明書の取得申請を出そう	92

4.5	DNS の設定をしよう	101
4.6	SSL 証明書をサーバに設置しよう	106
4.6.1	Windows で証明書と中間 CA 証明書をアップしよう	107
4.6.2	Mac で証明書と中間 CA 証明書をアップしよう	108
4.6.3	SSL 証明書と中間 CA 証明書を 1 ファイルにまとめよう	109
4.7	NGINX で HTTPS のバーチャルホストを作ろう	110
4.8	HTTPS でサイトを開いてみよう	111
	【コラム】ロードバランサーで SSL ターミネーションする方法もある	112
第 5 章	SSL/TLS について学ぼう	113
5.1	「サイトを HTTPS 化する」とは何か?	114
5.2	どんなサイトでも必ず HTTPS にしなきゃだめ?	114
5.3	HTTPS のままだと起きる「わるいこと」	114
5.3.1	サイトが「安全でない」と表示されてしまう	114
5.3.2	Wi-Fi スポットでセッションハイジャックされる恐れがある	115
5.3.3	相対的に検索順位が下がる	116
5.3.4	周りが HTTPS になるとリファラーが取れなくなる	116
5.4	HTTPS 化すると起きる「いいこと」	117
5.4.1	表示速度が上がる	117
5.4.2	Same Site 問題に対応できる	117
5.4.3	大事な情報のアタリが付けにくくなる	117
5.5	SSL/TLS とは?	118
5.5.1	SSL と TLS はどういう関係?	118
5.5.2	SSL イコール HTTPS ではない	118
5.6	SSL 証明書とは	119
5.6.1	SSL サーバ証明書と SSL クライアント証明書	119
5.6.2	SSL 証明書はどんな場面で使われている?	120
5.7	SSL 証明書は全然違う 2 種類の仕事をしている	121
5.7.1	ウェブサイトで送受信する情報を暗号化すること	121
5.7.2	ウェブページは 1 往復で表示されるわけじゃない	122
5.7.3	画像と CSS の指定が絶対パスだった	123
5.7.4	ウェブサイト運営者の身元を証明すること	124
5.7.5	ネットバンクの事例	124
5.8	認証局事業者の身元は誰が証明する?	126
5.8.1	身元保証の連鎖をつなぐ中間 CA 証明書とルート証明書	127

目次

5.9	SSL 証明書はどうしてあんなに値段に差があるの？	128
5.10	同じ「SSL 証明書」という名前でも 3 つの種類がある	129
5.10.1	3 つの違いは何か？	129
5.10.2	DV 証明書	129
5.10.3	EV 証明書と OV 証明書	130
5.10.4	さよならグリーンバー	131
5.10.5	ブラウザベンダーによる EV 証明書の扱いの変化	132
5.11	その他の証明書	132
5.11.1	中間証明書	132
5.11.2	クロスルート証明書	132
5.12	どの証明書を買えばいい？	132
5.12.1	ワイルドカード証明書	132
5.12.2	www ありにリダイレクトしたいだけなのに www なしの証明書 もいるの？	132
5.12.3	コモンネームが*.example.com の証明書は example.com で使え る？	132
5.12.4	SANs	132
5.12.5	Let'sEncrypt	132
5.13	CDN と証明書	132
5.13.1	CDN を使ったら古い端末でサイトが見られなくなった	132
5.13.2	同じサーバで複数サイトを HTTPS 化したら古い端末で別サイ トが表示された	132
5.13.3	SNI Server Name Indication	132
あとがき		133
PDF 版のダウンロード		134
Special Thanks:		134
レビュー		134
参考文献・ウェブサイト		134
著者紹介		137

第1章

Oracle Cloud のアカウントを作 ろう

この章では Oracle Cloud というクラウドでアカウントを作ります。

SSL を理解するには、実際に手を動かしてやってみるのがいちばんです。実際に SSL 証明書を取得して、HTTPS のサイトを作ってみましょう。

HTTPS でサイトを作るのに必要な材料は次の 3 つです。

- ウェブサーバ
- ドメイン名
- SSL 証明書

まずは 1 つめのウェブサーバを立てるため、アカウント作成から始めましょう。

1.1 サーバを立てる下準備

HTTPSでサイトを作るのに必要な材料は次の3つです。

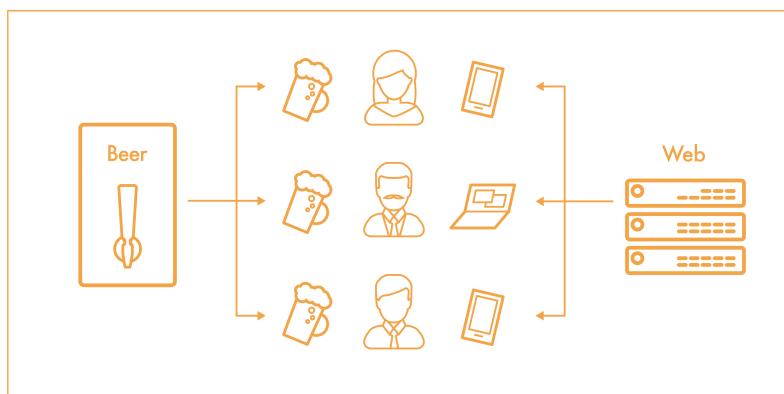
- ウェブサーバ
- ドメイン名
- SSL証明書

まずは1つめのウェブサーバを立てるための、下準備からはじめます。

1.1.1 サイトを作るのにどうしてサーバがいるの？

ところで、これからウェブサーバを立てようとしていますが…どうしてサイトを作りたいだけなのに、ウェブサーバが必要なのでしょう？

そもそもですが、サーバとはクライアントに対してサービスを提供するものです。居酒屋にあるビアサーバに「ビールをください」というリクエストを投げる…つまりコックを開の方へひねると、ビールというレスポンスが返ってきます。同様にあなたがブラウザでURLを入力したり、リンクをクリックしたりして、ウェブサーバに対して「ウェブページを見せてください」というリクエストを投げたら、ウェブページというレスポンスが返ってきます。（図1.1）



▲図1.1 ビアサーバもウェブサーバもリクエストしたらサービスが提供される

つまり、せっかくHTMLや画像でサイトのコンテンツを作っても、それを載せておくウェブサーバがなければ、サイトはあなたのパソコンの中でしか見られず、インターネット

トで公開できないのです。¹

というわけでウェブサイトを提供するために、まずはウェブサーバを立てましょう！

1.1.2 サーバを立てるにはお金が必要？

ウェブサイトを作るにはサーバが必要です。そしてサーバを立てるには、普通はお金がかかります。ですがオラクルがやっている「Oracle Cloud（オラクル クラウド）」というサービスなら、なんと有効期限なしでずっと無料で使える「Always Free」という枠があります。「Always Free」の範囲内であれば、サーバも無料で立てて使えるので今回はそれを使いましょう。

オラクルがやっているクラウド、と言われても、そもそもクラウドがなんだか分からないといまいちピンと来ないかもしれません。あなたが「ウェブサイト作りたいなあ…だからサーバが必要だ！」と思ったとき、**自分でサーバを買って自分で管理しなければいけないのがオンプレミスで、従量課金ですぐに使って性能や台数の増減も簡単にできるのがクラウドです。**

Oracle Cloud とはオラクルがやっているクラウドなので、ブラウザでぽちぽちとスペックを選んでいくだけで、すぐにサーバが使えます。

1.1.3 なんで AWS じゃなくて Oracle のクラウドを使うの？

クラウドは Oracle Cloud だけではありません。かの有名な AWS こと Amazon Web Services や、Google の Google Cloud Platform²、Microsoft の Azure（アジュール）³、その他にも国内クラウドとしてさくらインターネットがやっているさくらのクラウド⁴、お名前.com でお馴染み GMO グループの GMO クラウド⁵などたくさんあります。

2019年11月時点、クラウド市場では AWS がシェア約40%でトップを独走中⁶です。そのため仕事で AWS を使ったことがある、あるいはこれから使う予定だ、というエンジニアも多いと思います。

*¹ サーバについては、はじめようシリーズの2冊目、「AWSをはじめよう」の「CHAPTER1 インフラとサーバってなに？」で、より詳しく解説しています。仮想サーバと物理サーバ、クラウドとオンプレミス、ホストサーバとゲストサーバなどサーバ周りの用語をもう少し理解したい！という方はそちらも併せて読んでみるのがお勧めです

*² <https://cloud.google.com/>

*³ <https://azure.microsoft.com/ja-jp/>

*⁴ <https://cloud.sakura.ad.jp/>

*⁵ <https://www.gmocloud.com/>

*⁶ IaaS + PaaS クラウド市場、AWSの首位ゆるがず。AWS、Azure、Google、Alibaba の上位4社で市場の7割超。2019年第3四半期、Synergy Research Group — Publickey https://www.publickey1.jp/blog/19/iaaspaaawsazuregooglealibaba4720193synergy_research_group.html

しかし最近は、Alibaba Cloud や Tencent Cloud といった中国のクラウド事業者も追い上げを見せています。こうした新興のクラウドは、先に行く AWS を見て学んだ上で生まれてきているだけあって、よりスマートな作りになっているのがいいところです。

たくさんのクラウドがある中でどこを選ぶのか、その理由は、本来であれば使う人やその上で動かすサービスによって異なるはずです。あなたが動かしたいサービスには、一体どのクラウドが適しているのでしょうか？

本著では次の 2 つを目的としていますので、それに適した Oracle Cloud で学びを進めていきたいと思います。

- SSL 証明書を自分で取得して設置する一通りの流れを試したい
- お金をかけずに無料で試したい

1.2 Oracle Cloud でアカウント登録

まずは Oracle Cloud のアカウントを作りますので次の 2 つを用意してください。

- クレジットカード
- SMS 受信が可能な携帯電話（電話番号認証で使用するため）*7

なお Oracle Cloud を利用する際は、前述のとおり Always Free という無料枠*8があります。

1.2.1 無料でアカウントを作ろう

「Oracle Cloud 無料」で検索（図 1.2）したら、いちばん上の [Oracle Cloud Free Tier | Oracle 日本]*9をクリックします。

*7 ショートメッセージサービスの略。宛先に電話番号を指定してメッセージを送れるサービス

*8 期限なしでずっと無料ですが、無料で利用できる範囲は決まっていて、何をどれだけ使っても無料という訳ではありませんので注意してください。Always Free の他に、30 日間だけ有効な 300 ドル分の無償クレジットも付いてきますので、Always Free の範囲外のサービスはそちらで試せます。詳細は <https://www.oracle.com/jp/cloud/free/> を確認してください

*9 <https://www.oracle.com/jp/cloud/free/>

1.2 Oracle Cloud でアカウント登録



▲図 1.2 「Oracle Cloud 無料」で検索

Oracle Cloud Free Tier のページが表示されたら、[今すぐ始める（無償）] をクリックします。（図 1.3）

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.3 [今すぐ始める (無償)] をクリック

「Oracle Cloud へのサインアップ」と表示されたら、[電子メール・アドレス] と [国/地域] を入力して、使用条件を確認した上で [次] をクリックしましょう。(図 1.4) 後で分からなくなないように、登録した項目を表 1.1 にメモしておきましょう。

▼表 1.1 Oracle Cloud に登録した情報

項目	例	あなたが登録した情報
電子メール・アドレス	startdns.01@gmail.com	
国/地域	日本	



▲図 1.4 入力したら [次] をクリック

次は「アカウント詳細の入力」です。(表 1.2) 今回は仕事ではなく個人での利用ですので〔アカウント・タイプ〕は〔個人使用〕を選択してください。〔クラウド・アカウント名〕には任意のアカウント名を入力します。〔クラウド・アカウント名〕には英字小文字と数字のみ使えます。記号や英字大文字は使えないで注意してください。筆者は startdns01 にしました。この〔クラウド・アカウント名〕は、後で管理画面にサインインするときのアカウント URL になります。(図 1.5)

〔ホーム・リージョン〕は〔日本東部(東京)〕を選択してください。Oracle Cloud は世界の各地域にデータセンターを所有しており、サーバはそのデータセンターの中で元気に動いています。この〔ホーム・リージョン〕とは、**各地域の中でどこを使うか？を指定するものです**。ウェブサイトにアクセスするとき、パソコンのある場所からサーバまで物理的に距離が遠いと、それだけ通信にも時間がかかるって応答時間も遅くなります。日本国内向けにウェブサイトを開設する場合は、基本的にこの〔日本東部(東京)〕のリージョンを選びましょう。ただし Oracle Cloud のサービスによってはまだ東京リージョンが使えないものもあります。その場合は次点として「米国東部(アッシュバーン)」を選択してください。

▼表 1.2 Oracle Cloud に登録した情報

項目	例	あなたが登録した情報
アカウント・タイプ	個人使用	
クラウド・アカウント名	startdns01	
ホーム・リージョン	日本東部(東京)	

The screenshot shows the 'Account details input' screen. It has three main sections: 'Account type' (radio buttons for 'Corporate use' and 'Personal use', with 'Personal use' selected), 'Cloud account name' (text input field containing 'startdns01'), and 'Home region' (dropdown menu showing 'Japan East (Tokyo)'). A note at the bottom right of the region dropdown says: 'Always Freeのサービスが選択したリージョンで使用可能になりました。サービスの可用性については、[リージョン] を参照してください。' (The Always Free service you selected is available in the chosen region. For information on service availability, refer to [Region].)

▲図 1.5 [クラウド・アカウント名] には好きな名前を入力

続いて名前や住所を入力していきます。入力内容は日本語表記で構いません。個人利用なのですが「部門名」が必須であるため、ここでは「個人」と入力しておきましょう。「名」・「姓」・「部門名」・「住所」・「市区町村」・「都道府県」・「郵便番号」をすべて入力できましたか？（図 1.6）

The screenshot shows a registration form with several input fields highlighted by red boxes:

- 名 *: 猫村
- 姓 *: もち子
- 部門名 *: 借入
- 役職: (empty)
- 住所 *: 新宿四丁目1番6号
JR新宿ミライナタワー
- 市区町村 *: 新宿区
- 都道府県 *: TOKYO
- 郵便番号 *: 160-0022
- 国/地域: 日本

▲図 1.6 名前や住所を入力

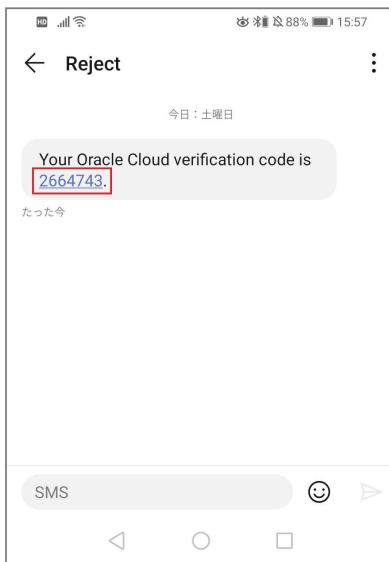
では最後に「モバイル番号」です。国番号は「日本(81)」を選択して、自分の携帯電話番号を入力します。このとき電話番号の先頭の 0 は不要です。例えば「090-〇〇〇〇-〇〇〇〇」という携帯電話番号であれば「90-〇〇〇〇-〇〇〇〇」と入力してください。携帯電話番号を入力したら【次: モバイル番号の確認】をクリックしてください。(図 1.7)

The screenshot shows a form for entering a mobile phone number:

- モバイル番号 *: 日本(81) [dropdown menu]
90 [input field]
- 下記のモバイル番号および電子メール startdn10@gmail.com に基づく検証コードがすでにある場合、[ここをクリックしてコードを確認してください。](#)
- 次: モバイル番号の確認 [button]
- サポートが必要ですか。ご連絡ください: [チャット・サポート](#)

▲図 1.7 携帯電話の番号を入力

数分以内に [Your Oracle Cloud verification code is 〇〇〇〇〇〇〇〇.] と書かれた SMS が届きます。(図 1.8)



▲図 1.8 コードの書かれた SMS が届いた

SMS で届いた「〇〇〇〇〇〇〇〇」の数字を [コード] に入力して、[コードの確認] をクリックします。(図 1.9)



▲図 1.9 SMS で届いた数字を [コード] に入力して [コードの確認] をクリック

1.2.2 〈トラブル〉 どうしても SMS が届かない！ そんなときは？

電話番号を入力したのに SMS が届かないときは、まず自分が契約している携帯キャリアの迷惑メール設定で、SMS をスパムとしてはじく設定をしていないか確認してみましょう。たとえば海外の事業者から送信された SMS を拒否する設定になっていたり、海外からの着信を拒否する設定になっていると、SMS が届かないことがあるようです。^{*10}

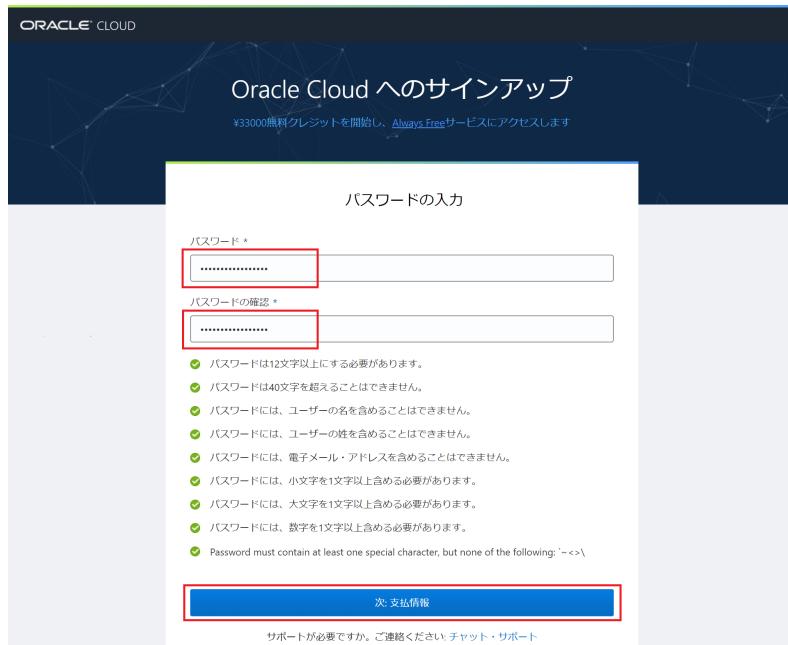
ちなみに筆者の場合は、特に設定変更をせず同じ番号で 2 回試してみたのですが、最初は届かず、もう 1 回試してようやく届きました。

迷惑メールの設定を確認して何回か試して、それでも SMS が届かなかったら、ページ下部の [サポートが必要ですか。ご連絡ください: チャット・サポート] からサポートにチャットで問い合わせてみましょう。残念ながら英語でしか対応してもらえませんが、"I am trying to register on Oracle Cloud. But I can't receive SMS. What should I do?" (アカウント登録しようとしてるけど SMS が届かないの、どうしたらいい？) という感じで聞いてみると、「じゃあ登録情報をこのチャットで教えて。そうしたらこちらでコードを発行して、このチャットで伝えてあげる」(意訳) という感じでサポートしてもらえます。

1.2.3 パスワードと支払情報の登録

正しいコードが入力できたら、[パスワードの入力] が表示されます。[パスワード] と [パスワードの確認] を入力して、[次: 支払情報] をクリックします。(図 1.10)

^{*10} 「Oracle Cloud の SMS は海外の事業者から届く」という確証がある訳ではないです。あくまで SMS が届かないときによくある話と思ってください



▲図 1.10 パスワードを入力して [次: 支払情報] をクリック

パスワードを入力すると、今度は「支払情報」のページが表示されます。(図 1.11) 繰り返しあ伝えしているとおり、Oracle Cloud には Always Free という無料枠があります。本著では基本的にこの無料枠の範囲内で Oracle Cloud を使っていくつもりですが、それでもクレジットカードは登録しておく必要があります。

なおページに記載されているとおり、この後、管理画面で「アカウントのアップグレード」という作業をしない限り、請求は発生しませんので安心してカード情報を登録してください。[クレジット・カード詳細の追加] をクリックします。



▲図 1.11 [クレジット・カード詳細の追加] をクリック

[ご注文者様情報] はそのまま変更不要です。[カード情報] の [カードの種類] を選択し、[カードの番号]・[有効期限]・[CVN] を入力したら [Finish] をクリックします。
(図 1.12)*¹¹

*¹¹ Oracle Cloud では、クレジットカード登録時に「1 ドル認証」と呼ばれる認証方法で、そのクレジットカードが決済可能かをチェックしています。クレジットカードによってはこの 1 ドル認証を不審な決済と判断して通さないため、それによってエラーが発生することがあります。その場合は別のクレジットカードで試すか、Oracle Cloud のチャット・サポートで問い合わせてみてください

カード情報 ▲

カードの種類 *

Visa Mastercard
 Amex JCB

カードの番号 *

有効期限 *

CVN *

このコードは、クレジットカードの裏面または表面に印字されている3桁または4桁の番号です。

▲図 1.12 カード情報を入力して [Finish]

[クレジット・カード詳細をご提供いただきありがとうございます。] と表示（図 1.13）されたら、支払い情報の登録は完了です。Oracle Cloud の Service Agreement^{*12}を確認した上で、チェックボックスにチェックを入れて、[サインアップの完了] をクリックします。

*12 <https://www.oracle.com/goto/oraclecsa-jp-en>

1.2 Oracle Cloud でアカウント登録



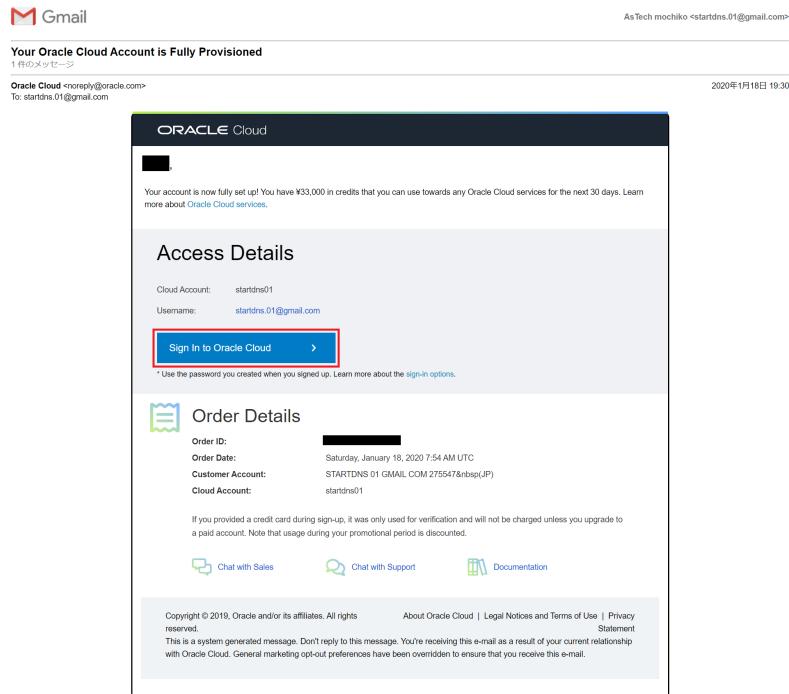
▲図 1.13 チェックを入れて [サインアップの完了] をクリック

これでアカウント登録の手続きはおしまいです。[アカウントの設定が完了するまでお待ちください。] と表示（図 1.14）されます。準備が整うとサインイン画面にリダイレクトされますが、この [アカウントの設定が完了するまでお待ちください。] の画面でかなり時間がかかるので、一度ブラウザを閉じてしまって構いません。頑張った自分を褒めて、一旦休憩にしましょう。



▲図 1.14 アカウント登録の手続きはおしまい

数時間後^{*13}、[Your Oracle Cloud Account is Fully Provisioned] という件名で、準備完了を知らせるメールが届きます。メールの [Sign In to Oracle Cloud] をクリックしましょう。(図 1.15)



▲図 1.15 数時間後、準備完了を知らせるメールが届く

1.2.4 Oracle Cloud のコンソールにサインイン

メールの [Sign In to Oracle Cloud] をクリックすると、コンソールへのサインイン^{*14}画面が表示されます。(図 1.16) [ユーザー名] には先ほど登録したメールアドレスを入力します。^{*15} [パスワード] を入力して、[サイン・イン] をクリックしてください。

*13 筆者の場合は、メールが届くまで 2 時間半かかりました

*14 日本語だとログインの方が馴染みがあるかも知れませんが、サインインはログインと同じ意味です

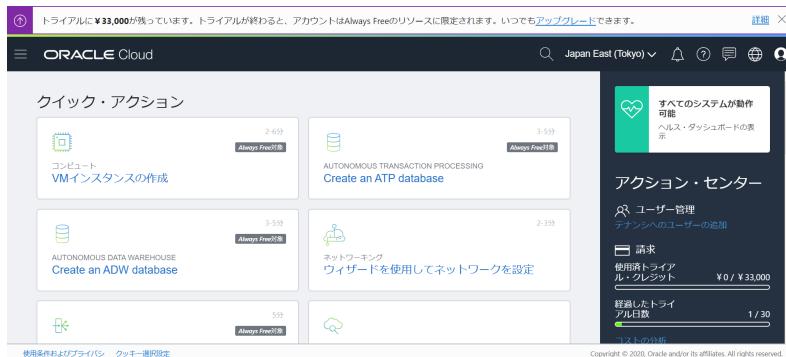
*15 メールにも書いてありますが、ここでの [ユーザー名] とは [クラウド・アカウント名] (筆者の場合は startdns01) ではなく、[メールアドレス] のことです。紛らわしいのでご注意ください

1.2 Oracle Cloud でアカウント登録



▲図 1.16 [ユーザー名] と [パスワード] を入力して [サイン・イン]

おめでとうございます！ コンソールにサインインできました。



▲図 1.17 コンソールにサインインできた！

なお今後、コンソールにサインインしたくなったら、いちいち Oracle Cloud からのメールを探してリンクを踏む必要はありません。まずは Oracle のトップページ^{*16}を開いて、右上の人マークから [クラウドにサインイン] をクリックしましょう。

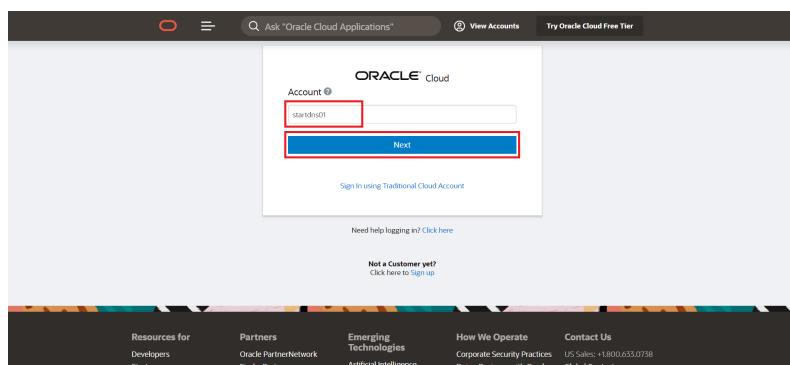
*16 <https://www.oracle.com/jp/>

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.18 右上の人マークから [クラウドにサインイン] をクリック

サインインのページ^{*17}で [Account] の欄にクラウド・アカウント名^{*18}を入力して [Next] をクリックすれば、メールのリンクを踏んだときと同じ [サイン・イン] のページにたどり着けます。あとは同じように [ユーザー名] にはメールアドレスを、[パスワード] にはパスワードを入力して、[サイン・イン] をクリックするだけです。



▲図 1.19 [Account] の欄にクラウド・アカウント名を入力して [Next] をクリック

*17 <https://www.oracle.com/cloud/sign-in.html>

*18 筆者の場合は startdns01 です。アカウント登録時に、あなたの [クラウド・アカウント名] をメモしているはずですでの、数ページ戻って確認してみましょう

第2章

Oracle Cloud でサーバを立てよう

この章では実際に Oracle Cloud でサーバを立てます。
インフラエンジニアのお仕事体験みたいできっと楽しいですよ！

2.1 事前準備

2.1.1 Windows で RLogin をインストールする

Windows のパソコンを使っている方は、サーバを立てる前に「ターミナル」と呼ばれる黒い画面のソフトをインストールしておきましょう。サーバに接続するときにはこのターミナルを使うのですが、ターミナルのソフトには色々な種類があります。

- RLogin (<http://nanno.dip.jp/softlib/man/rlogin/>)
- Poderosa (<https://ja.poderosa-terminal.com/>)
- Tera Term (<https://ja.osdn.net/projects/ttssh2/>)
- PuTTYjp (<http://hp.vector.co.jp/authors/VA024651/PuTTYkj.html>)



▲図 2.1 RLogin

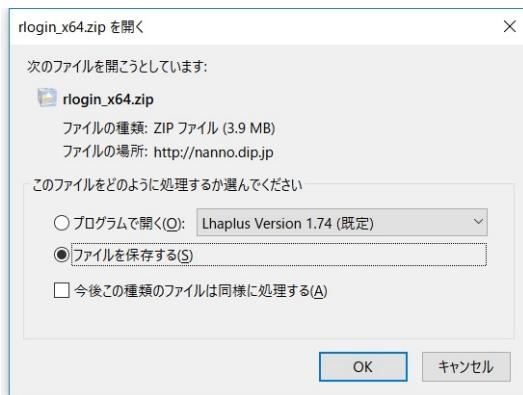
本著ではいちばん上の RLogin (図 2.1) を使って説明していくので、特にこだわりがなければ RLogin を使うことをお勧めします。RLogin の「実行プログラム (64bit)^{*1}」(図 2.2) の URL、http://nanno.dip.jp/softlib/program/rlogin_x64.zip をクリックしてください。

^{*1} もしパソコンの Windows が 32bit 版だった場合は「実行プログラム (32bit)」の URL をクリックしてください。



▲図 2.2 「実行プログラム (64bit)」の URL をクリックしてダウンロード

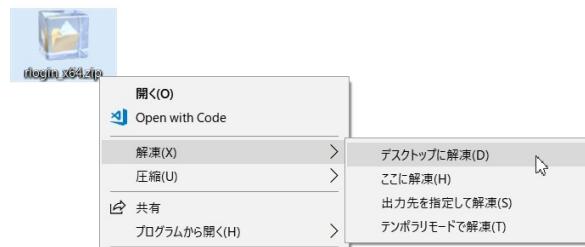
ダウンロードした ZIP ファイルを保存（図 2.3）します。保存場所はどこでも構いませんが、後でどこに置いたか分からなくなりそうな人はデスクトップに保存しておきましょう。



▲図 2.3 「ファイルを保存する」でパソコンに保存

デスクトップの ZIP ファイル (rlogin_x64.zip) を右クリック（図 2.4）して、[解凍>デスクトップに解凍]*2をクリックします。

*2 ZIP ファイルを右クリックしても「解凍」が見当たらないときは、圧縮・解凍の定番ソフトである Lhaplus をインストールしましょう。 <https://forest.watch.impress.co.jp/library/software/lhaplus/>



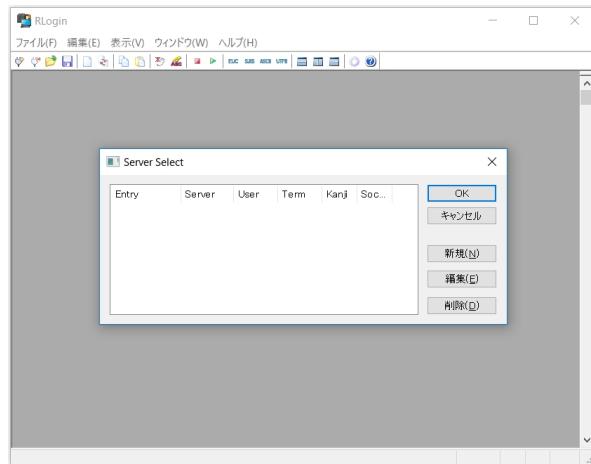
▲図 2.4 ZIP ファイルを右クリックして解凍>デスクトップに解凍

解凍したら、デスクトップにできた「rlogin_x64」というフォルダの中にある「RLogin.exe」*3（図 2.5）をダブルクリックすれば RLogin が起動（図 2.6）します。



▲図 2.5 RLogin.exe をダブルクリック

*3 フォルダの中に RLogin はあるけど RLogin.exe なんて見当たらない…という場合、ファイルの拡張子が非表示になっています。この後も拡張子を含めてファイル名を確認する場面が何度かでできますので、表示されていない人は「拡張子 表示」で Google 検索して、拡張子が表示されるように設定変更しておきましょう。

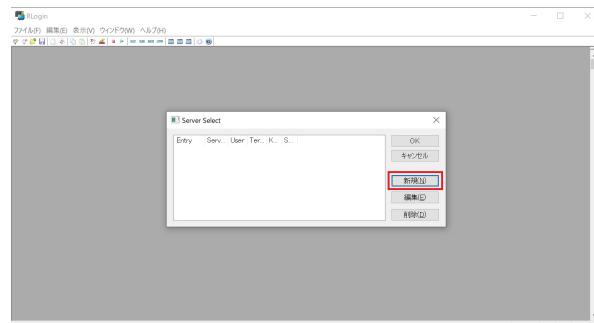


▲図 2.6 RLogin が起動した

これで RLogin のインストールは完了です。

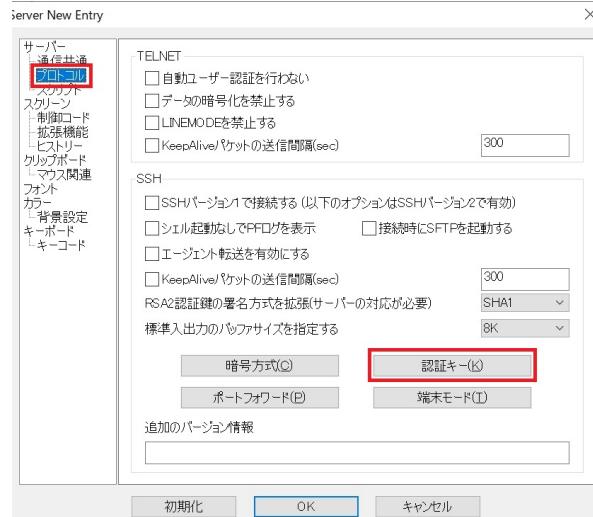
2.1.2 Windows で SSH のキーペア（秘密鍵・公開鍵）を作成する

Windows の方は、続いて起動した RLogin で [新規 (N)] をクリックします。



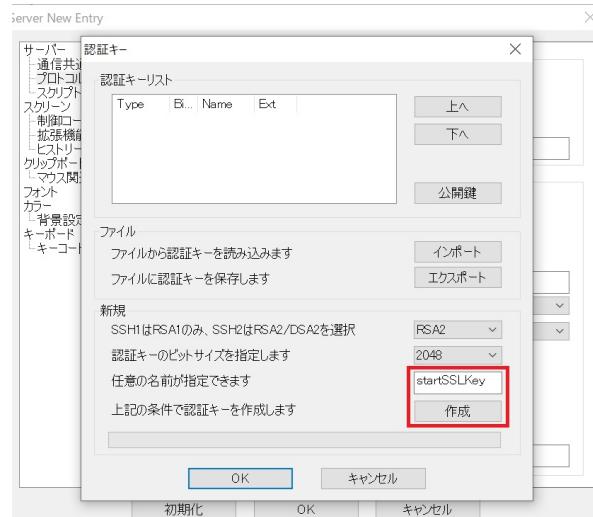
▲図 2.7 [新規 (N)] をクリック

左メニューの [プロトコル] を選択して、[認証キー (K)] をクリックします。



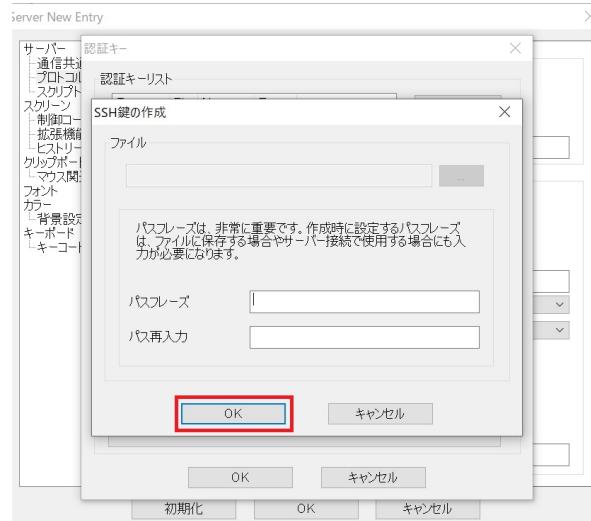
▲図 2.8 [プロトコル] を選択して [認証キー (K)] をクリック

[任意の名前が指定できます] に [startSSLKey] を入力して、[作成] をクリックします。



▲図 2.9 [startSSLKey] を入力して [作成] をクリック

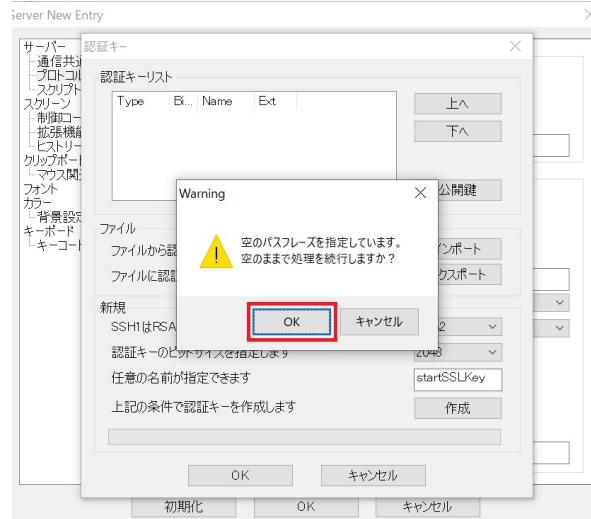
[パスフレーズ] と [パス再入力] には何も入力せず、[OK] をクリックします。^{*4}



▲図 2.10 何も入力せず [OK] をクリック

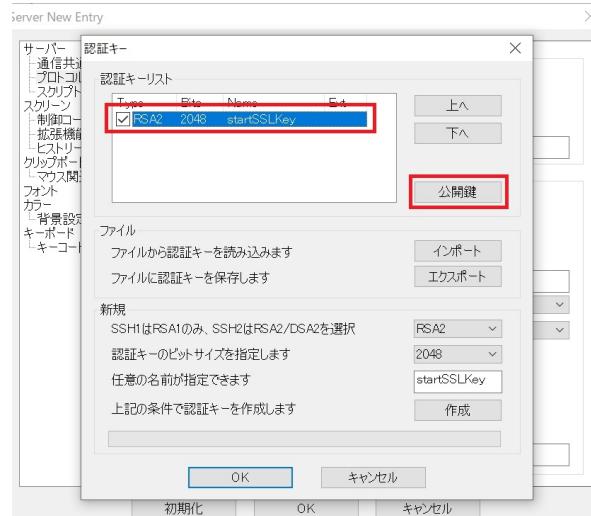
[空のパスフレーズを指定しています。空のままで処理を続行しますか？] と表示されますが、そのまま [OK] をクリックします。

^{*4} 「p@\$sw0rd」 や「@dm1ni\$trat0r」 のように、ひとつの単語でできているのがパスワードです。それに対して「This 1s P@ss Phrase.」のように空白を挟んだ文章（フレーズ）で構成されているのがパスフレーズです



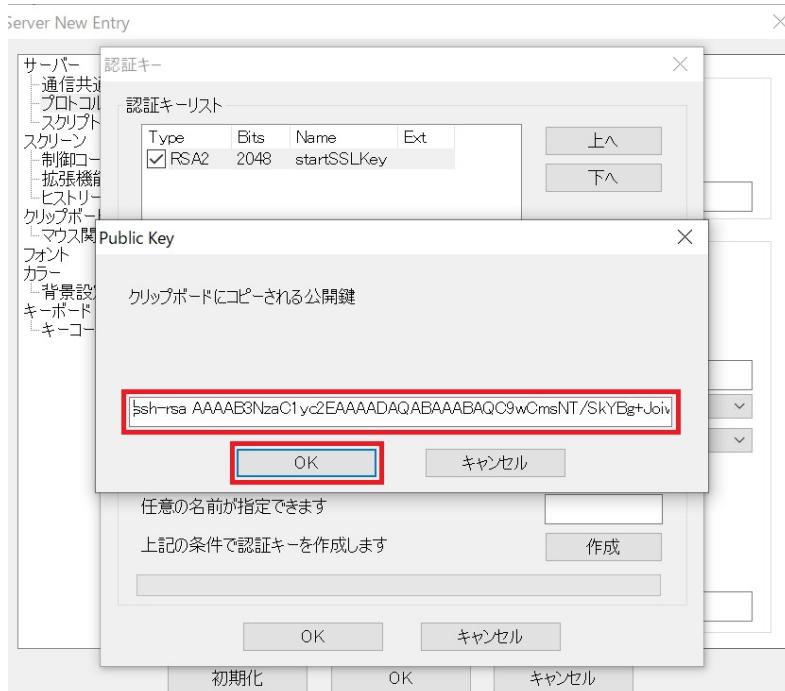
▲図 2.11 [OK] をクリック

【認証キーリスト】に、いま作った【startSSLKey】が表示されたら、キーペア（秘密鍵・公開鍵）が無事できています。【公開鍵】をクリックしてください。（図 2.5）



▲図 2.12 【startSSLKey】が表示されたら【公開鍵】をクリック

この後すぐに使いますので、[クリップボードにコピーされる公開鍵] で表示された公開鍵（ssh-rsa から始まる文字列）をまるごとコピーして、メモ帳などにペーストしておきましょう。公開鍵をメモしたら [OK] をクリックして閉じます。



▲図 2.13 表示された公開鍵（文字列）はまるごとコピーしてメモ帳にペーストしておこう

あとは [キャンセル] を繰り返しクリックして、起動中の RLogin はいったん閉じてしまって構いません。RLogin は、後でサーバへ入るときに使いますので、デスクトップの「rlogin_x64」フォルダと、その中にある「RLogin.exe」をごみ箱へ捨てないように注意してください。メモした公開鍵も無くさないようご注意ください。

【コラム】SSH の秘密鍵にパスフレーズは設定すべき？

秘密鍵に [パスフレーズ] を設定しておくと、鍵を使って SSH でサーバに入ろうとしたとき、「鍵を発動するにはパスフレーズを叫べ…！」という感じでパスフレーズを聞かれます。

つまり、もしあなたの秘密鍵が盗まれて誰かに勝手に使われそうになっても、パスフレーズを設定していれば鍵の悪用が防げます。スマホ本体が盗まれてしまっても、パスワードが分からなければロック画面が解除できず、勝手に使えないのと同じです。

これは「あなたは何を持っているのか」「あなたは何を知っているのか」「あなたは誰なのか」という複数の要素の中から、2つを用いることで認証の強度を高める「二要素認証」と呼ばれる考え方です。「あなたは秘密鍵を持っている」「あなたはパスフレーズを知っている」という2つの要素を組み合わせることで、単要素での認証よりも強度が高まります。ちなみに「あなたが誰なのか」は指紋認証や顔認証ですね。

ですが「パスワード認証じゃなくて鍵認証なのに、やっぱりパスフレーズが要るの…？」という具合に、初心者を混乱に陥れやすいので、本著では秘密鍵をパスフレーズなしで作って使います。

パスフレーズは「設定していれば絶対に安心！」というものではありませんが、上記の理由から、本来であれば設定した方がいいものです。後で「やっぱり設定しておこう」と思ったら、一度作成した秘密鍵に後からパスフレーズを設定することも可能です。

2.1.3 Mac でターミナルを準備する

Mac を使っている方は、最初から「ターミナル」(図 2.14) というソフトがインストールされていますのでそちらを利用しましょう。



▲図 2.14 最初からインストールされている「ターミナル」を使おう

ターミナルがどこにあるのか分からぬときは、Mac の画面で右上にある虫眼鏡のマークをクリックして、Spotlight で「ターミナル」と検索（図 2.15）すれば起動できます。



▲図 2.15 どこにあるのか分からなかつたら Spotlight で「ターミナル」と検索

2.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する

Mac の方は、ターミナルで次のコマンドを実行してください。⁵

```
$ ssh-keygen -f ~/Desktop/startSSLKey
```

すると次のように、パスフレーズの入力待ち状態になります。何も入力せずに、2回 Enter を押してください。

```
$ ssh-keygen -f ~/Desktop/startSSLKey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): ←何も入力せずに Enter
Enter same passphrase again: ←何も入力せずに Enter
```

次のように表示されたらキーペア（秘密鍵・公開鍵）の作成は完了です。

```
$ ssh-keygen -f ~/Desktop/startSSLKey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mochikoAsTech/Desktop/startSSLKey.
Your public key has been saved in /home/mochikoAsTech/Desktop/startSSLKey.pub.
The key fingerprint is:
a2:52:43:dd:70:5d:a8:4f:77:47:ca:f9:69:79:14:48 mochikoAsTech@ghana
The key's randomart image is:
+--[ RSA 2048]----+
|       . .. ooE. |
|       . + o . . |
|       . . . . +. |
|       . . . = o |
|       o . So . . +o |
|       . o . . +o |
|       . . . . |
|       .           |
+-----+
```

ホームディレクトリに秘密鍵（startSSLKey）と、公開鍵（startSSLKey.pub）ができる

⁵ ssh-keygen コマンドは名前のとおり、SSH の鍵（key）を生成（generate）するコマンドです。-f オプションでは、生成する鍵のファイル名を指定しています。～（チルダ）はホームディレクトリを表しますので、-f ~/Desktop/startSSLKey は「/Users/<ユーザ名>/Desktop」のフォルダの中に「startSSLKey」という名前の鍵を作って、という意味です

あがっているはずです。cat（キャット）コマンド^{*6}で公開鍵を表示してみましょう。

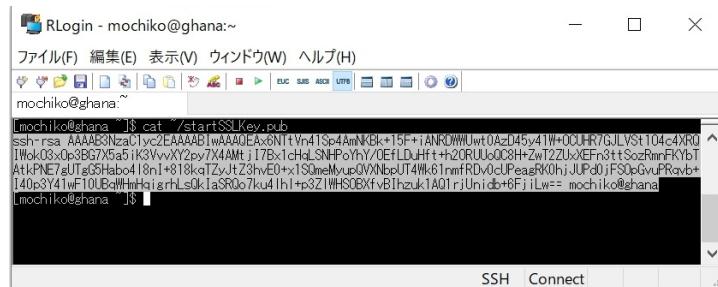
```
$ cat ~/startSSLKey.pub
ssh-rsa AAAAB3NzaC1yc2EAAQEAx0NtVn4TSpc4AmNKB+15f+iANR0WWUwt0AzD45y41W+00UHR7GJLVS1t04c4XRQ
Iwok03xOp3B67X5a1K3VyyXY2pY7X4AMfj17Bx1chLsNHPoYhY/0EfLDUhft+h20RUuOC8H+ZwT2ZuXEFn3ttSozRmnFKYbt
AtkPNE/gUlg5Habo418n1+818kgq1ZyJtZ3hvU0+x1SuMeMyupQVNnbU14Wk61nmfRDvJcUFeaghk0hjJUPd0)FSUpGvuRavb+
[40p3Y41wF10UBdIffhajgrhLs0k1aSR067ku4lh1+pSZIHSDOBxfvB1hzukTA01rjUnidb+6FjiLw== mochiko@ghana]
```

この後すぐに使いますので、表示された公開鍵（ssh-rsa から始まる文字列）をまるごとコピーして、メモ帳などにペーストしておきましょう。

以上で事前準備は完了です。お待たせしました。いよいよサーバを立てましょう。

【コラム】ターミナルでコピー＆ペーストするには？

ターミナルで表示されている内容をコピーしたいときは、コピーしたい部分をマウスで選択するだけです。（図 2.16）選択してから Ctrl+c を押す必要はありません。



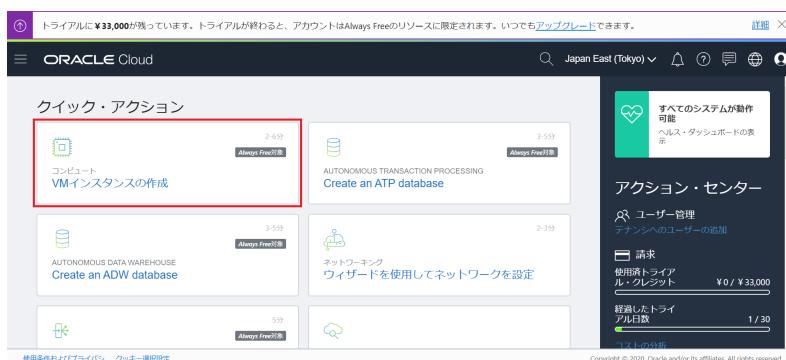
▲図 2.16 マウスで選択するだけでコピーできる

逆にコピーした内容をターミナルへペーストしたいときはターミナル上で**右クリックするだけ**です。ターミナルのソフトにもよりますが、基本的に Ctrl+p は使えないで注意してください。

^{*6} cat は猫ではなく「conCATenate files and print on the standard output」の略です

2.2 コンピュートでサーバを立てる

それでは下準備ができたので、Oracle Cloud のコンソールに戻ってサーバを立てましょう。コンソールにサインインしたら、[VM インスタンスの作成] をクリック（図 2.17）します。ちなみに Oracle Cloud にはデータベース管理やストレージなど、さまざまなサービスがありますが、クラウドサーバや物理サーバなどのサーバが立てられるサービスは「コンピュート」と呼ばれています。そして Oracle Cloud ではサーバのことを、**インスタンス**と呼びます。ここから先でインスタンスと書いてあつたら「サーバのことだな」とだと思ってください。



▲図 2.17 [VM インスタンスの作成] をクリック

[インスタンスの命名] に [startSSLInstance] と入力します。（図 2.18）その下の [オペレーティング・システムまたはイメージ・ソースを選択します] は、何も変更せずそのまま構いません。



▲図 2.18 [インスタンスの命名] に [startSSLInstance] と入力

2.2.1 OS は Oracle Linux 7.7 を使おう

パソコンには OS という基本ソフトが入っていて、Word や Excel、Chrome といったソフトはその OS の上で動いています。皆さんのパソコンにも「Windows 10」や「Mac OS X Lion」などの OS が入っていますよね。

そしてパソコンと同じようにサーバにも「Linux」や「Windows Server」といったサーバ用の OS があります。サーバを立てるときには Linux を選択することが多いのですが、この Linux の中にもさらに「RHEL (Red Hat Enterprise Linux)」や「CentOS」、「Ubuntu」などいろいろなディストリビューション（種類）があります。

本著では、OS はデフォルトの [Oracle Linux 7.7] を使用します。Oracle Linux なら Oracle Cloud のツールがあらかじめ入っていますので、**Oracle Linuxでサーバを立てるときはOSはOracle Linuxにすることをお勧めします**。Oracle Linux は Red Hat 系のディストリビューションですので、RHEL や CentOS のサーバを使ったことがある方なら違和感なく使えると思います。

2020 年 1 月時点で、Oracle Linux には次の 2 種類があります。

- Oracle Linux 6.10
- Oracle Linux 7.7

名前のとおり、Oracle Linux 6.10 は CentOS 6 と同じ RHEL6 系、Oracle Linux 7.7 は CentOS 7 と同じ RHEL7 系なので、使い勝手はほぼ同じです。

2.2.2 作っておいた SSH の公開鍵を設置しよう

さらに下に進んで [SSH キーの追加] は、[SSH キーの貼付け] を選択して、そこに先ほどメモしておいた公開鍵をペーストします。公開鍵は改行を含まず、先頭の「ssh-rsa」から末尾の「<ユーザ名>@<ホスト名>」のようなコメントまでで、まるごと 1 行です。(図 2.19)



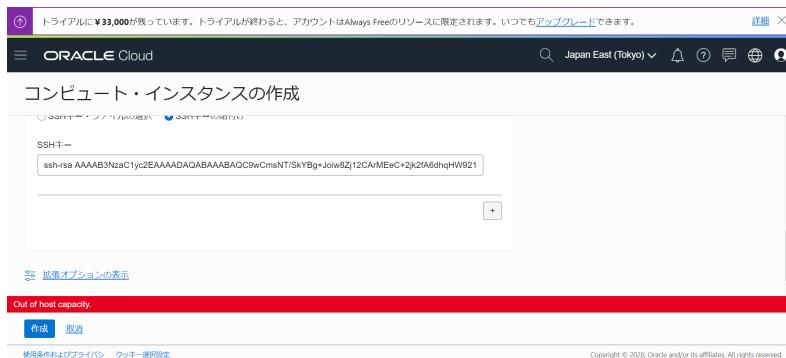
▲図 2.19 [SSH キーの貼付け] を選択してメモしておいた公開鍵をペースト

公開鍵をペーストしたら [作成] をクリックします。

2.2.3 <トラブル> "Out of host capacity."が起きたらどうすればいい？

さて、元気よく [作成] をクリックしたのに、真っ赤な [Out of host capacity.] が表示されてしまった…という方が一定数いらっしゃると思います。大丈夫、あなたは悪くありません。いま理由を説明するので落ち着いてください。「そんなの表示されなかったよ？」という方は、このコラムは読み飛ばして、この先の「サーバが起動するまで待とう」までジャンプしてください。

2.2 コンピュートでサーバを立てる



▲図 2.20 "Out of host capacity."と表示されて何も起きない！

"Out of host capacity."は、直訳すると「ホスト容量が不足しています」という意味ですが、ホストってなんでしょう？

あなたがいま Oracle Cloud で立てようとしたサーバは、家でいうと「一軒家」ではなく、マンションの 101 号室や 403 号室のような「各部屋」にあたります。このときマンションの建物をホストサーバ、各部屋をゲストサーバと呼びます。



▲図 2.21 マンションの建物をホストサーバ、各部屋をゲストサーバと呼ぶ

「ホストの容量が不足している」ということは…つまり、あなたが Oracle Cloud の無料マンションに入居しようとしたら、「ごめんね、無料マンションは大人気でいま空き部屋がないの」と断られてしまった、という状況なのです。

Oracle Cloud の Always Free は有効期限なしでずっと無料で使える、とても魅力的なサービスです。そのため Oracle Cloud 側も定期的に新築マンションを追加しているもの

の、定期的にリソース不足に陥ってはこういう状況になるようです。

この"Out of host capacity."が発生してしまった場合、次のどちらかが起きてホストの容量不足が解消しない限り、Always Free の枠でサーバは立てられません。

- 自分以外のユーザーがサーバを解約してリソースを開放する
- Oracle Cloud がリソースを増やす

ですが、Always Free とは別に、我々には 30 日間だけ有効な\$300 の無償クレジットが与えられています。たとえ無料マンションが満室でも、有料マンションなら空きがあります。30 日経ったら消えてしまう\$300 のお小遣いを握りしめたら、次の方法で有料マンションのお部屋を借りにいきましょう！

2.2.4 Always Free ではなく無償クレジットの枠でサーバを立てよう

次の手順は、"Out of host capacity."が表示された人だけ実施してください。

もともと選択していたのは[Always Free 対象]のマークが付いた[VM.Standard.E2.1.Micro(仮想マシン)] という種類のサーバでした。"Out of host capacity."を回避するため、少し上にスクロールして [シェイプ、ネットワークおよびストレージ・オプションの表示] をクリックしましょう。(図 2.22)



▲図 2.22 [シェイプ、ネットワークおよびストレージ・オプションの表示] をクリック

Oracle Cloud では、サーバスペックごとに「シェイプ」という区分があります。^{*7} インスタンスのシェイプを、いま選択されている [VM.Standard.E2.1.Micro] から変更した

^{*7} シェイプとはサーバスペックごとの区分のことです。AWS のインスタンスタイプと同じものだと思ってください

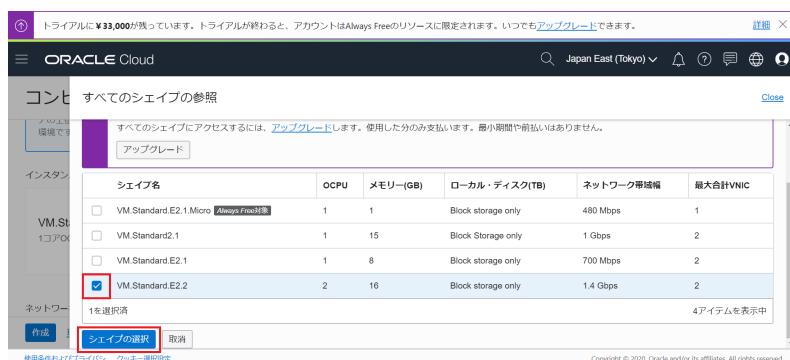
2.2 コンピュートでサーバを立てる

いので [シェイプの変更] をクリックしてください。



▲図 2.23 [シェイプの変更] をクリック

OCPU^{*8}が 2、メモリが 16GB の [VM.Standard.E2.2]^{*9}にチェックを入れて、[シェイプの選択] をクリックしましょう。



▲図 2.24 「VM.Standard.E2.2」に変更して [シェイプの選択] をクリック

それ以外は何も変更せずに、いちばん下の [作成] をクリックします。

*8 OCPU は Oracle Compute Units の略で、ごく簡単に言うと物理 CPU です。OCPU (物理 CPU) 1つは、vCPU (仮想 CPU) 2つに相当しますので、もし「AWS の EC2 で vCPU が 4 のサーバを使っている。同等スペックのサーバを用意してほしい」と頼まれたら、Oracle Cloud では OCPU が 2 のシェイプを選べば大丈夫です。単純に数字だけで比較して、OCPU が 4 のシェイプを選ぶと CPU のスペックがいままでの倍になってしまいますので注意してください

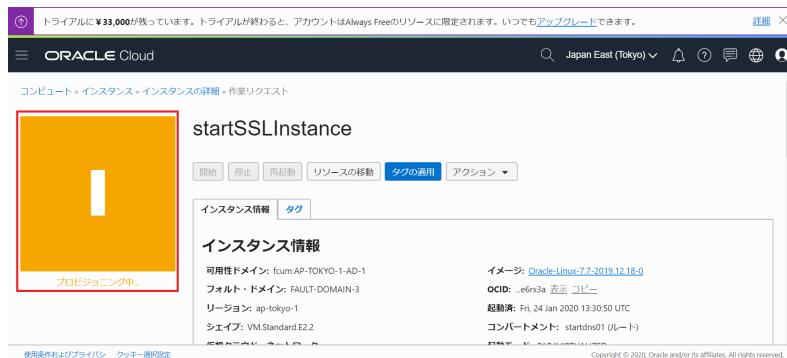
*9 シェイプの名前は、まず接頭辞が「VM」なら仮想サーバ (Virtual Machine)、「BM」なら物理サーバ (Bare Metal) を表しています。その後ろの単語は「Standard」(汎用) や「DenseIO」(高密度 IO) といった特徴、3番目の「E2」や「3」はシェイプの世代、最後の「2」や「8」は OCPU の数を表しています



▲図 2.25 [作成] をクリック

2.2.5 サーバが起動するまで待とう

オレンジ色で「[プロビジョニング中...]」と表示されたら、サーバが用意されるまでそのまま数分待ちましょう。



▲図 2.26 「[プロビジョニング中...]」と表示されたら数分待つ

サーバができあがると、表示が緑色の「[実行中]」に変わります。おめでとうございます！ これでサーバが立てられました！



▲図 2.27 「実行中」に変わった！ サーバが立てられた！

【コラム】Oracle Cloud のコンピュートの金額計算方法

ところで、いま立てた「VM.Standard.E2.2」をまるまる 1 ヶ月使ったら、一体いくら分になるのでしょうか？ うっかり \$300 を超えてしまわないか、ちょっと心配なので計算してみましょう。

コンピュートの価格表^{*10}を見てみると、[VM.Standard.E2.2] は [\$.03]^{*11}と書いてあります。これは [Pay as You Go (OCPU Per Hour)] と書いてあるとおり、1OCPU につき 1 時間あたりかかる金額です。^{*12}

「VM.Standard.E2.2」は OCPU が 2 なので、\$.03*2 で 1 時間あたり \$.06 かかることが分かります。1 ヶ月を 744 時間 (24 時間*31 日) として、\$.06*744 時間で \$44.64 です。

「VM.Standard.E2.2」を 1 台立てたくらいでは、\$300 の無償クレジットを使い切ることはないので安心しましょう。ちなみに Oracle Cloud では \$1 は 120 円換算^{*13}なので、日本円だと 5356.8 円ですね。

^{*12} <https://www.oracle.com/jp/cloud/compute/pricing.html>

^{*13} 2020 年 1 月時点の金額

^{*14} ちなみに AWS は、同スペックのサーバでもリージョンごとに価格が異なりますが、Oracle Cloud はどここのリージョンでも同一の価格です

^{*15} \$1 を 120 円で換算すると \$44.64*120 円で 5356.8 円です

【コラム】Oracle Cloud と AWS はどっちが安い？

Oracle Cloud は他のクラウドに比べて価格が安いのが特徴のひとつです。どれくらい安いのか、同じスペックのサーバで AWS と比較してみましょう。

例えば同スペックの VM.Standard.E2.1 (Oracle Cloud) と m5.large (AWS) を比較すると、Oracle Cloud の価格は AWS の 4 分の 1 以下です。(表 2.1)

▼表 2.1 Oracle Cloud と AWS の価格比較

	Oracle Cloud	AWS
インスタンスの種類	VM.Standard.E2.1	m5.large
CPU	OCPU:1 (vCPU:2相当)	vCPU:2
メモリ	8GB	8GB
1 時間あたり	\$0.03	\$0.124
月額	2678.4 円	11070.72 円

シェアトップを独走する AWS に対して、後発は勝つためにコスト面や性能面でそれぞれ大きなメリットを打ち出してきています。AWS が最適なのであれば AWS を選択すべきですが、「みんなが使っているから」というだけ理由で、あまり深く考えずに AWS を使っているのであれば、他のクラウドにも目を向けてみることを筆者はお勧めします。

2.2.6 接続先となるサーバの IP アドレス

無事にサーバが「実行中」になったら、接続先となるサーバの IP アドレスを確認してみましょう。

先ほど作成したインスタンス [startSSLInstance] の、[プライマリ VNIC 情報] (図 2.28) にある [パブリック IP アドレス] をメモ (表 2.2) してください。

2.2 コンピュートでサーバを立てる

The screenshot shows the Oracle Cloud Infrastructure (OCI) console with the instance details page for an instance named 'startstest'. The 'Primary VNIC Information' section is highlighted, showing the Public IP address '140.238.33.51'.

例	パブリック IP アドレス
140.238.33.51	

▲図 2.28 [プライマリ VNIC 情報] の [パブリック IP アドレス] をメモしておこう

▼表 2.2 インスタンスの [パブリック IP アドレス]

例	パブリック IP アドレス
140.238.33.51	

それではメモした IP アドレスを使ってサーバに入ってみましょう。

第3章

ウェブサーバの設定をしよう

この章ではウェブサーバの設定を行ないます。

3.1 サーバに SSH でログインしよう

では立てたばかりのサーバに、SSH でログインしてみましょう。

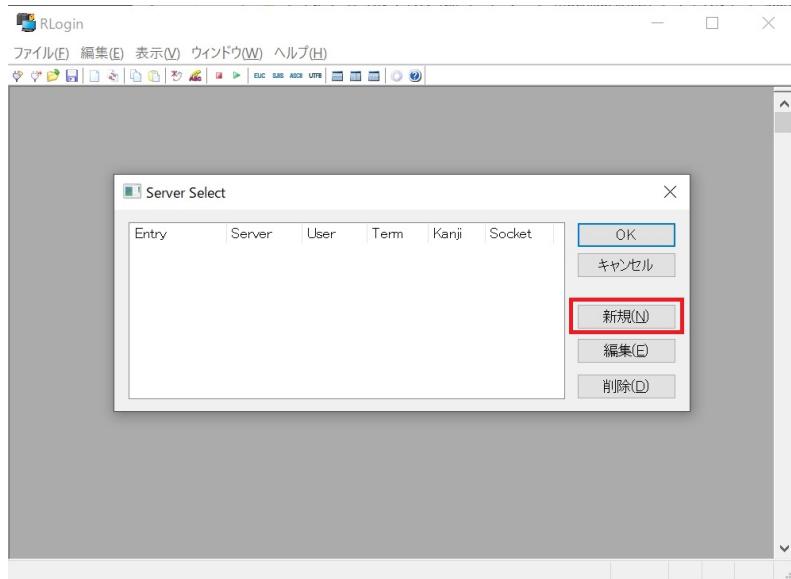
3.1.1 Windows の RLogin を使ってサーバに入ってみよう

Windows のパソコンを使っている方は、デスクトップの [rlogin_x64] というフォルダの中にある [RLogin.exe]（図 3.1）をダブルクリックして RLogin を起動（図 3.2）してください。起動したら [新規] をクリックします。



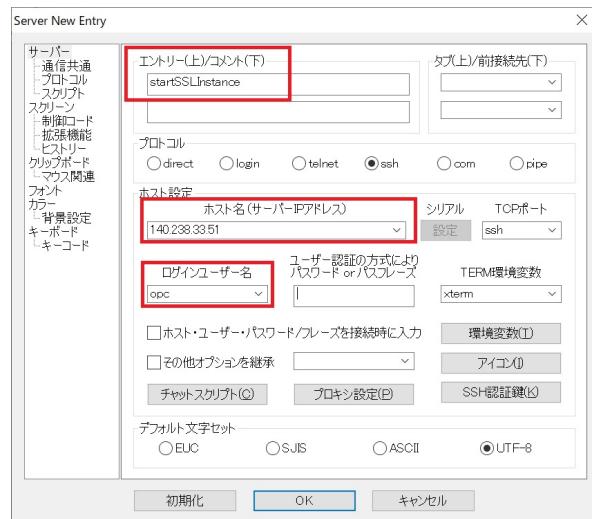
▲図 3.1 RLogin.exe をダブルクリック

3.1 サーバに SSH でログインしよう



▲図 3.2 RLogin が起動したら [新規] をクリック

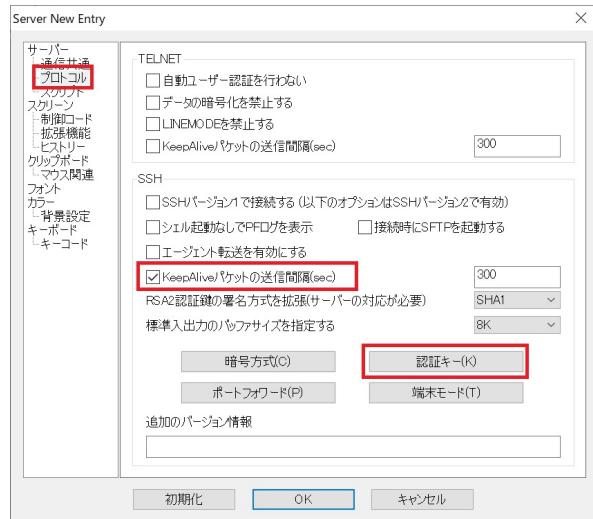
初めに [エントリー (上) /コメント (下)] の上に [startSSLInstance] と入力します。続いて [ホスト名 (サーバー IP アドレス)] に先ほどメモした [パブリック IP アドレス] を入力 (図 3.3) します。[ログインユーザー名] には [opc] と入力してください。opc というのは Oracle Linux のインスタンスを作成すると、最初から存在しているデフォルトユーザです。



▲図 3.3 [ホスト名 (サーバー IP アドレス)] と [ログインユーザー名] を入力

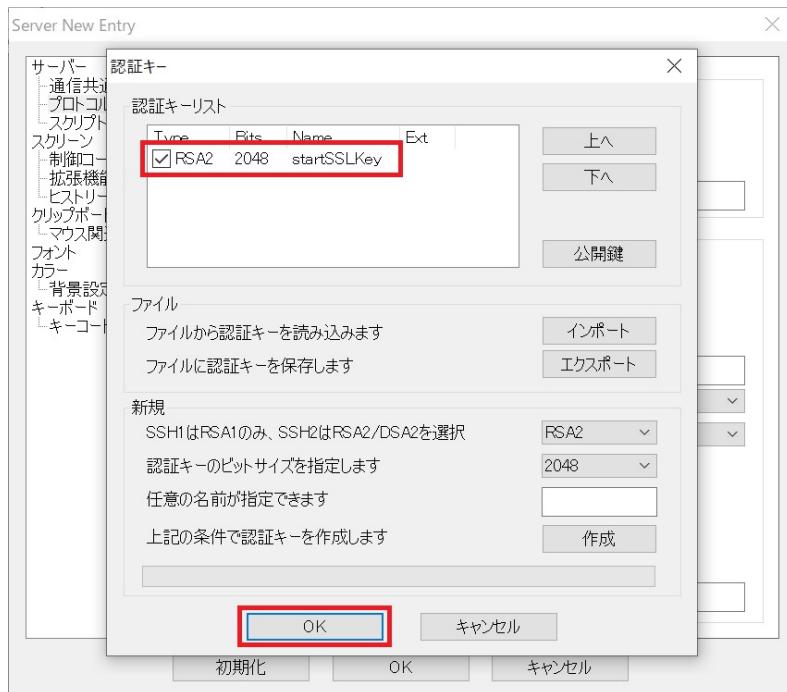
次に左メニューで [プロトコル] を選択（図 3.4）したら、[KeepAlive パケットの送信間隔 (sec)] にチェックを入れておきます。これを設定しておくとターミナルをしばらく放っておいても接続が勝手に切れません。続いて [認証キー] をクリックします。

3.1 サーバに SSH でログインしよう



▲図 3.4 [KeepAlive パケットの送信間隔 (sec)] にチェックを入れて [認証キー] をクリック

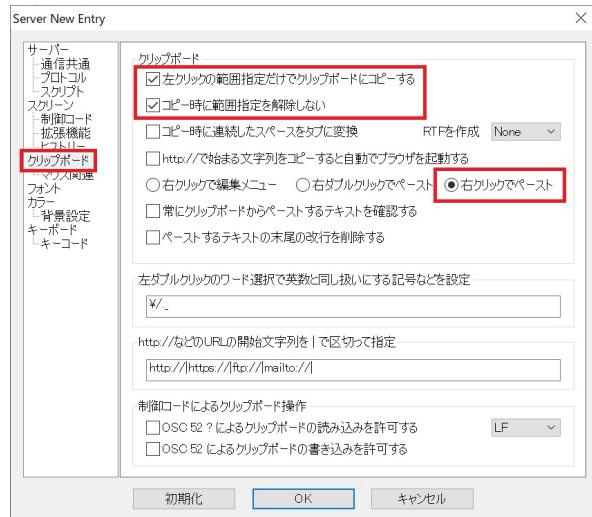
[認証キー] リストで [startSSLKey] にチェックが入っていることを確認（図 3.5）します。これは「ログインするときにこの鍵を使います」というリストです。チェックが入っていたら [OK] をクリックして閉じて構いません。



▲図 3.5 「startSSLKey」にチェックが入っていることを確認

続いて左メニューで「クリップボード」を選択（図 3.6）したら、「左クリックの範囲指定だけでクリップボードにコピーする」と「コピー時に範囲指定を解除しない」にチェックを入れて「右クリックでペースト」を選択します。

3.1 サーバに SSH でログインしよう



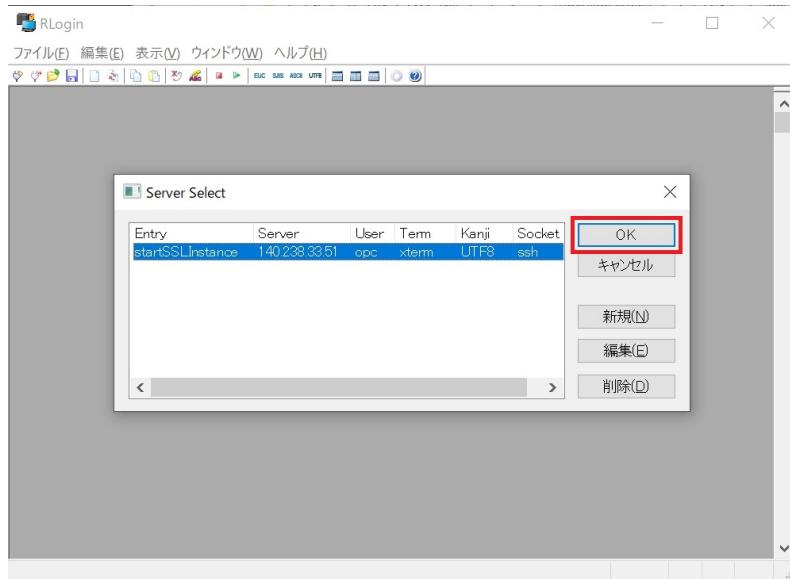
▲図 3.6 右クリックや左クリックの設定

次に左メニューで「フォント」を選択（図 3.7）したら、文字セットを「UTF-8」に変更します。すべて設定できたら「OK」をクリックしてください。



▲図 3.7 文字セットを「UTF-8」に変更

設定が保存できたら「OK」をクリック（図 3.8）してください。



▲図 3.8 設定が保存できたら「OK」をクリック

すると初回のみ、この「公開鍵の確認」が表示（図 3.9）されます。これは「初めて入るサーバだけど信頼していいですか？本当に接続しますか？」と聞かれているので、「接続する」をクリックしてください。サーバにはそれぞれフィンガープリントという固有の指紋があるため、下部の「この公開鍵を信頼するリストに保存する」にチェックが入っていれば RLogin が覚えていてくれて、次回以降は「これは前に信頼していいって言われたサーバだ！」と判断してそのまま接続させてくれます。



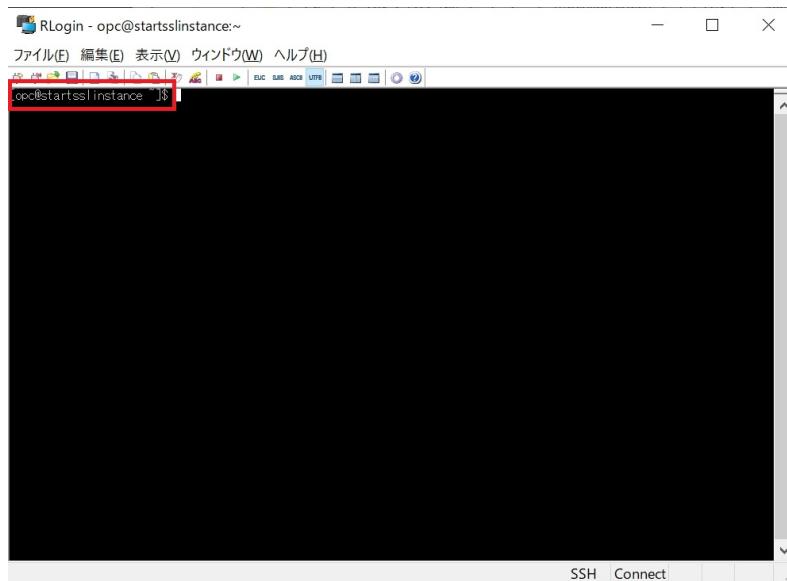
▲図 3.9 「公開鍵の確認」が表示されたら「接続する」をクリック

続いて「信頼するホスト鍵のリストを更新しますか？」と聞かれたら「はい」をクリック（図 3.10）してください。



▲図 3.10 「信頼するホスト鍵のリストを更新しますか？」と表示されたら「はい」をクリック

「opc@startsslinstance」と表示（図 3.11）されたら無事サーバに入っています。SSHでのログイン成功、おめでとうございます！



▲図 3.11 「opc@startsslinstance」と表示されたら成功！

もし「opc@startsslinstance」と表示されず、代わりに「SSH2 User Auth Failure "publickey,ssapi-keyex,ssapi-with-mic" Status=1004 Send Disconnect Message... ssapi-with-mic」というようなエラーメッセージが表示（図 3.12）されてしまったら、これは「鍵がない人は入れないよ！」とお断りされている状態です。【認証キー】リストで【startSSLKey】にチェックが入っていないものと思われますので【認証キー】の設定を確認してみてください。



▲図 3.12 このエラーが表示されたら【認証キー】を確認しよう

「接続済みの呼び出し先が一定の時間を過ぎても正しく応答しなかったため、接続できませんでした。」というエラーメッセージが表示（図 3.13）されてしまった場合は、「ホスト名（サーバー IP アドレス）」に書いた「パブリック IP アドレス」が間違っているものと思われます。「ホスト名（サーバー IP アドレス）」の IP アドレスを確認してみてください。



▲図 3.13 このエラーが表示されたら「ホスト名（サーバー IP アドレス）」の IP アドレスを確認しよう

3.1.2 Mac のターミナルを使ってサーバに入ってみよう

Mac を使っている方は、ターミナル（図 3.14）を起動してください。

3.1 サーバに SSH でログインしよう



▲図 3.14 最初からインストールされている「ターミナル」を使おう

ターミナルがどこにあるのか分からぬときは、Mac の画面で右上にある虫眼鏡のマークをクリックして、Spotlight で「ターミナル」と検索（図 3.15）すれば起動できます。



▲図 3.15 どこにあるのか分からなかつたら Spotlight で「ターミナル」と検索

そして開いたターミナルで次の文字を入力して Return キーを押します。これはサーバに入るときに使う鍵をオーナー以外が使えないよう、chmod というコマンドで読み書き権限を厳しくしています。この作業は最初の1回だけ構いません。もし「startSSLKey」を保存した場所がデスクトップ以外の場合は適宜書き換えてください。

```
$ chmod 600 ~/Desktop/startSSLKey
```

続いてターミナルで次の文字を入力したら再び Return キーを押します。「パブリック IP アドレス」の部分は先ほどメモした「パブリック IP アドレス」に書き換えてください。-i オプションは「サーバにはこの鍵を使って入ります」という意味ですので、「startSSLKey」を保存した場所がデスクトップ以外だった場合はこちらも適宜書き換えてください。

```
$ ssh opc@パブリック IP アドレス -i ~/Desktop/startSSLKey
```

初回のみ次のようなメッセージが表示されますが、これは「初めてに入るサーバだけど信頼していいですか？本当に接続しますか？」と聞かれていますので、「yes」と打ってReturnキーを押してください。するとMacはちゃんとこのサーバのことを覚えてくれて、次回以降は「これは前に信頼していいって言われたサーバだ！」と判断してそのまま接続させてくれます。

```
Are you sure you want to continue connecting (yes/no)?
```

「opc@startsslinstance」と表示されたら無事サーバに入っています。おめでとうございます！

今後はいまやったのと同じやり方をそのまま繰り返せばサーバにログインできます。ドメイン名というとどうしても「ブラウザで入力してサイトを見るときに使うもの」というイメージがありますが、「名前からIPアドレスが引けるもの」なのでこういう使い方もできるのです。

3.2 ターミナルでサーバを操作・設定してみよう

ようやくサーバを入れたので、ここからはターミナルの基本的な操作を試してみましょう。

3.2.1 プロンプトとは？

では黒い画面で何回か Enter キー（あるいは Return キー）を押してみましょう。（図 3.16）普通に改行されますよね。



▲図 3.16 Enter キーを押すと改行されて、プロンプトが常に表示されている

このとき左側にずっと出ている次のような表示は「プロンプト」といって、ログインしているユーザ名やサーバの名前などが表示されています。

```
[opc@startsslinstance ~]$
```

プロンプトを見るといまは「opc」という一般ユーザであることが分かります。これからサーバに色々な設定をしたいのですが、一般ユーザだと権限がないので「root」という全権限をもったユーザになりましょう。

```
$ sudo su -
```

と書いて Enter キーを押すと root になります。（図 3.17）「\$」はプロンプトを表していますので入力しないでください。root になれたままた何回か Enter キーを押して改行してみましょう。



▲図 3.17 sudo su -を書いて Enter キーを押すと root になれる

いちばん左側に出ているプロンプトが次のように変化しましたか？

```
[root@startsslinstance ~]#
```

ユーザ名が「opc」から「root」に変わりました。それからいちばん右の部分も「\$」から「#」に変わっています。プロンプトは**一般ユーザだと「\$」で全権を持っているrootだと「#」**という表示になります。今後は「このコマンドを root で実行してください」のように実行ユーザを詳しく書くことはしませんので、例として書いてある部分のプロンプトが「\$」だったら opc のような一般ユーザで実行、「#」だったら root で実行するんだ、と思ってください。例に「\$」や「#」が書いてあってもターミナルで「\$」や「#」を自分で入力する必要はありません。

3.2.2 コマンドは失敗したときだけエラーを吐く

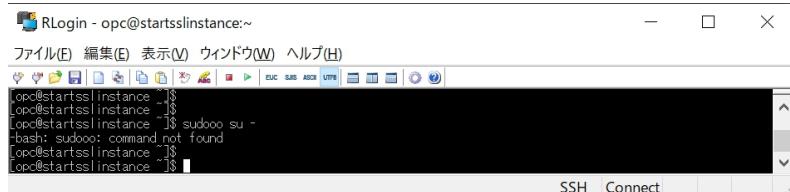
前述の「sudo su -」のようなものを「コマンド」と呼びます。コマンドとはサーバに対して「あれをして」「これをして」と頼む命令のようなものです。

サーバでコマンドを打った場合、基本的に**上手くいったときは何も言わないで失敗したときだけエラーを吐きます**。ですのでコマンドを打った時に何も表示されなくても不安にならなくて大丈夫です。

先ほどの「私を root にして！」という命令である「sudo su -」も、上手くいってちゃんと root になれたのでメッセージは一切出ていないですよね。これが「sudo」を打ち間違えて、こんな風に実行するとどうなるでしょう？

```
$ sudooo su -
```

「sudooo なんてコマンドは見つからなかったよ」というエラーメッセージ（図 3.18）が表示されました。



▲図 3.18 「sudooo: command not found」というエラーが表示された

このように何か失敗したときだけエラーが出ます。英語でエラーが出るとそれだけでパニックになってしまいますが、落ち着いてゆっくり読めば「sudooo: command not found…ああ、sudooo っていうコマンドが見つかりませんでした、って書いてある」と判読できると思います。エラーが出たら声に出してゆっくり読んでみましょう。

3.2.3 ターミナルを閉じたいとき

もう今日の勉強は終わり！ サーバとの接続を切ってターミナルを閉じたい、というときは exit (イグジット) というコマンドを叩きます。

```
# exit
```

root になっているときに exit を叩くと opc に戻れます。そして opc で再び exit を叩くと、サーバの接続を切ってターミナルを閉じることができます。

```
$ exit
```

exit をせずに右上の赤い×を押してウィンドウを閉じるのは、電話を切るときに通話オフのボタンを押さずに電話線を引っ張くような乱暴な切り方なのでお勧めしません。

3.3 NGINX をインストールしよう

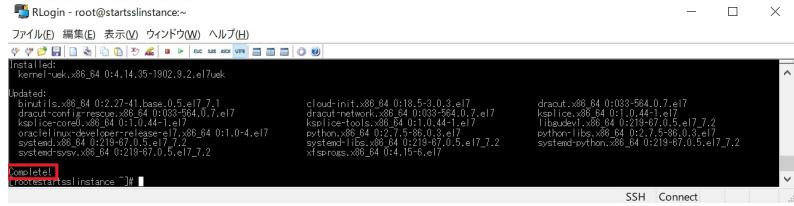
それでは必要なミドルウェアをインストールしていきましょう。最初に root になっておいてください。インストールするときは yum (ヤム) というコマンドを使います。

```
$ sudo su -
```

第3章 ウェブサーバの設定をしよう

先ずは yum で色々アップデートしておきましょう。Windows アップデートみたいなものです。画面にたくさん文字が流れ、少し時間がかかりますが、最後に [Complete!] と表示されたら問題なく完了しています。(図 3.19) ちなみに -y オプションは YES を意味するオプションです。-y オプションをつけないで実行すると「これとこれを更新するけどいい？ ダウンロードサイズとインストールサイズはこれくらいだよ」という確認が表示されて、y と入力して Enter キーを押さないと更新されません。

```
# yum update -y
```



▲図 3.19 最後に [Complete!] と表示されたらアップデートは完了

続いて NGINX^{*1}を入れます。2020 年 1 月現在の安定バージョン^{*2}である 1.16 系をインストールしたいので、yum のリポジトリ（どこから NGINX をダウンロードしていくか）に NGINX 公式を追加しましょう。

```
# rpm -ivh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.centos.1-1.el7.noarch.rpm
# cat /etc/yum.repos.d/nginx.repo

[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/7/$basearch/
gpgcheck=0
enabled=1
```

*1 NGINX と書いて「えんじんえっくす」と読みます。ウェブサーバのミドルウェアの中で NGINX は順調にシェアを伸ばし、2019 年 4 月にとうとう Apache を抜いてシェア 1 位になりました。<https://news.mynavi.jp/article/20190424-813722/>

*2 サーバに入っている NGINX のバージョンがいくつなのか？ という情報は大切です。今後、あなたが NGINX の設定ファイルを書こうと思って調べたとき、例えば 1.12 系の設定方法を参考にしてしまうと、1.16 系の NGINX の環境では上手く動かない可能性があります。

yum で NGINX をインストールします。

```
# yum install -y nginx
```

[Complete!] と表示されたらインストール完了です。バージョン情報を表示することで、ちゃんとインストールされたか確認してみましょう。

```
# nginx -v  
nginx version: nginx/1.16.1 ←バージョン情報が表示されればインストールできている
```

ちょっと分かりにくいかも知れませんが、パソコンに Microsoft Excel をインストールしたら「表計算というサービスが提供できるパソコン」になるのと同じで、サーバにこの NGINX をインストールすると「リクエストに対してウェブページを返すサービスが提供できるサーバ」、つまりウェブサーバになります。今回は NGINX を入れましたが、ウェブサーバのミドルウェアは他にも Apache をはじめとして色々な種類があります。

インストールが終わったので、サーバを再起動した場合も NGINX が自動で立ち上がりてくるよう、自動起動の設定もオンにしておきましょう。systemctl コマンドで、NGINX の自動起動を有効 (enable) にします。

```
# systemctl enable nginx  
# systemctl is-enabled nginx  
enabled ←有効になったことを確認
```

3.4 Firewalld で HTTP と HTTPS を許可しよう

続いてサーバの中で動いているファイアウォールの設定を変更します。まずは現状、何がファイアウォールを通れるようになっているのか確認してみましょう。

```
# firewall-cmd --list-services  
dhcpcv6-client ssh
```

dhcpcv6-client と ssh は通つていいけれど、それ以外は誰であろうと通さないぞ！ という設定になっています。このままではブラウザでサイトを見ようとしても、ウェブページを返してくれるはずの NGINX までリクエストが届きません。そこで次のように、許

可対象に http と https を追加して、変更が反映されるよう再読み込みします。それぞれ [success] と表示されたら成功しています。

```
# firewall-cmd --add-service=http --permanent  
success  
  
# firewall-cmd --add-service=https --permanent  
success  
  
# firewall-cmd --reload  
success
```

これでファイアウォールの設定が変更できたので、もう一度、誰がファイアウォールを通してもらえるのか確認してみましょう。http と https が追加されていれば問題ありません。

```
# firewall-cmd --list-services  
dhcpcv6-client http https ssh
```

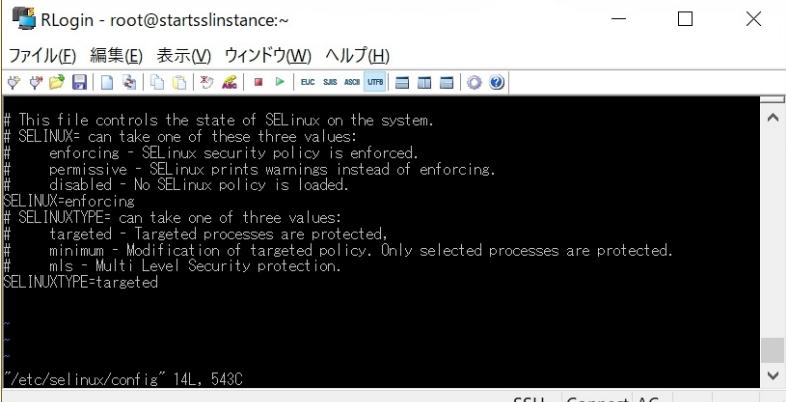
3.5 SELinux を無効にしておこう

SELinux を無効化 (disabled) します。

```
# getenforce  
Enforcing ←起動直後は有効になっている  
  
# vi /etc/selinux/config
```

vi (ブイアイ) はテキストファイルを編集するためのコマンドです。vi コマンドでファイルを開くと、最初は次のような「閲覧モード」の画面（図 3.20）が表示されます。閲覧モードは「見るだけ」なので編集ができません。

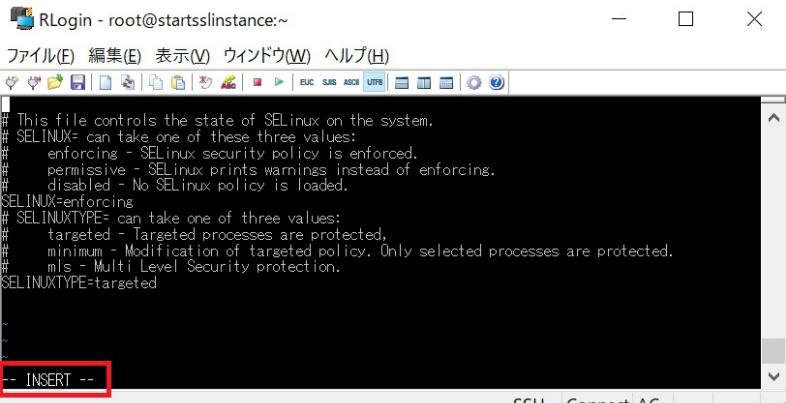
3.5 SELinux を無効にしておこう



```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

▲図 3.20 vi コマンドでファイルを開いた

この状態で i (アイ) を押すと「編集モード」*3に変わります。(図 3.21) 左下に「-- INSERT --」と表示されていたら「編集モード」です。

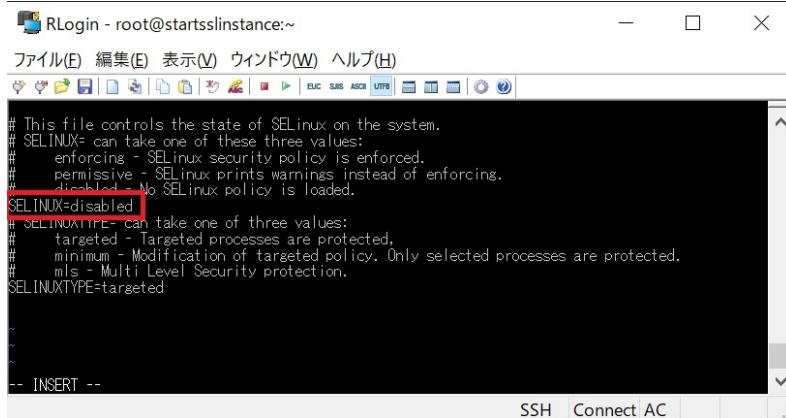


```
# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

▲図 3.21 i (アイ) を押すと「-- INSERT --」と表示される「編集モード」になった

「編集モード」になるとファイルが編集できるようになります。それでは「SELINUX=enforcing」を「SELINUX=disabled」(図 3.22) に書き換えてください。

*3 ここでは初心者の方でも直感的に分かるよう「閲覧モード」「編集モード」と呼んでいますが、正しくは「ノーマルモード」「インサートモード」です。



RLogin - root@startsslinstance:~

ファイル(F) 編集(E) 表示(V) ウィンドウ(W) ヘルプ(H)

This file controls the state of SELinux on the system.
SELINUX= can take one of these three values:
enforcing - SELinux security policy is enforced.
permissive - SELinux prints warnings instead of enforcing.
disabled - No SELinux policy is loaded.
SELINUX=disabled
SELINUXTYPE= can take one of three values:
targeted - Targeted processes are protected,
minimum - Modification of targeted policy. Only selected processes are protected.
mls - Multi Level Security protection.
SELINUXTYPE=targeted

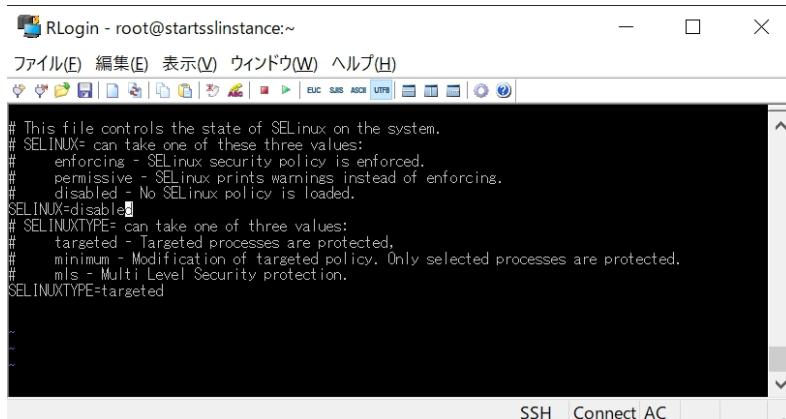
-- INSERT --

SSH Connect AC

This screenshot shows a terminal window titled "RLogin - root@startsslinstance:~". The window contains a text editor displaying the SELinux configuration file. The line "SELINUX=disabled" is highlighted with a red box. Below the text area, there is a status bar with "SSH" and "Connect AC" buttons.

▲図 3.22 「SELINUX=enforcing」を「SELINUX=disabled」に書き換える

「編集モード」のままだと保存ができないので書き終わったら ESC キーを押します。すると左下の「-- INSERT --」が消えて再び「閲覧モード」になります。(図 3.23)



RLogin - root@startsslinstance:~

ファイル(F) 編集(E) 表示(V) ウィンドウ(W) ヘルプ(H)

This file controls the state of SELinux on the system.
SELINUX= can take one of these three values:
enforcing - SELinux security policy is enforced.
permissive - SELinux prints warnings instead of enforcing.
disabled - No SELinux policy is loaded.
SELINUX=disabled
SELINUXTYPE= can take one of three values:
targeted - Targeted processes are protected,
minimum - Modification of targeted policy. Only selected processes are protected.
mls - Multi Level Security protection.
SELINUXTYPE=targeted

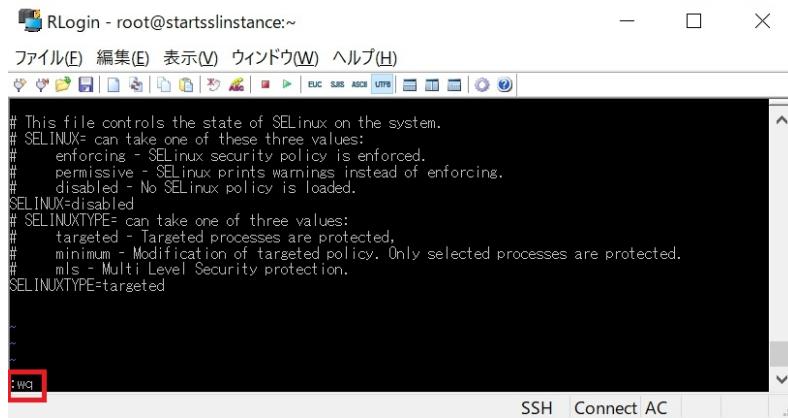
SSH Connect AC

This screenshot shows the same terminal window as in Figure 3.22, but the "INSERT" mode indicator at the bottom left has disappeared, indicating that the user has pressed the ESC key to switch back to "View Mode".

▲図 3.23 ESC を押すと左下の「-- INSERT --」が消えて再び「閲覧モード」になる

「閲覧モード」に戻ったら「:wq」^{*4}と入力して Enter キーを押せば変更が保存されます。(図 3.24)

^{*4} 書き込んで (write)、抜ける (quit) という命令なので wq です。



▲図 3.24 「:wq」と入力して Enter キーを押せば保存される

色々やっているうちになんだか誤が分からなくなってしまって「いまの全部なかったことにしたい！ 取り合えず vi からいったん抜けたい！」と思ったときは、ESC キーを押して「:q!」⁵と入力して Enter キーを押すと変更を保存せずに抜けることができます。

編集できたら cat (キャット) コマンドでファイルの中身を確認してみましょう。

```
# cat /etc/selinux/config

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE= can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

以上で SELinux を無効化する設定は終わりです。

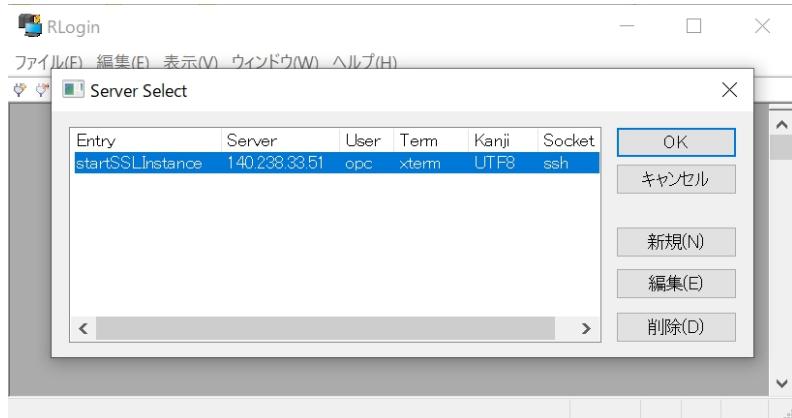
3.6 OS を再起動してみよう

変更した設定を反映させるため reboot (リブート) コマンドでサーバを再起動しておきましょう。

⁵ 保存せずに強制終了 (quit!) という命令なので q!です。

```
# reboot
```

SSH の接続も切れてしまいますが、割とすぐに再起動しますので再度 RLogin やターミナルで接続（図 3.25）してみてください。今度はさっきと同じ設定でそのまま接続できるはずです。



▲図 3.25 さっきと同じ設定で接続してみよう

以上で「サーバを立てる」という作業はおしまいです。

3.6.1 ターミナルはなんのためにある？

ターミナルで yum や vi を叩いてサーバの設定を色々してきましたが、ここで「結局、ターミナルって何なの？」という振り返りをしておきましょう。

ターミナルはサーバを操作するための画面です。

皆さんのがパソコン使うときはモニタに表示された画面を見ながらキーボードとマウスを使って「フォルダを開いて先週作った Word ファイルを探す」とか「Word ファイルを開いて今週の報告書を書く」というような操作をするとと思います。フォルダを開くときは「ダブルクリック」をして、書いた内容を保存するときは「上書き保存する」ボタンを押しますよね。

サーバも同じです。サーバを使うときは「ターミナル」という画面を開いて操作します。ディレクトリ^{*6}を開いて移動するときはダブルクリックの代わりに cd^{*7}というコマ

*6 Linux ではフォルダのことをディレクトリと呼びます。

*7 change directory の略。

ンドを叩いて移動しますし、ディレクトリの中を見るとときもダブルクリックでフォルダを開く代わりに ls コマンドを叩いて見ます。

皆さんのがいま使っている Windows や Mac といった「パソコン」だったらマウスやキーボードを使ってアイコンやボタンを見ながら操作できますが、サーバは基本的にこの真っ黒な「ターミナル」で文字を打って操作します。パソコンのときはダブルクリックやボタンを押す、という形で伝えていた命令がすべてコマンドに置き換わっていると思ってください。

パソコンもないのにマウスやキーボードだけあっても意味が無いように、ターミナルもそれ単体では何もできません。操作対象であるサーバがあつて初めて役に立つ道具なのです。

ちなみにターミナルは背景の色も文字の色も好きに変えられます。どうしても「黒い画面怖い！」という感覚が抜けない人は、ピンクとかオレンジとか好きな色にしてみましょう。^{*8}

まとめるとターミナルとはサーバを操作するための画面で、操作するときにはコマンドという命令を使います。

3.7 なぜかサイトが見られない

ウェブサーバも立てたし、SELinux はとめたし、サーバの中のファイアウォールに穴も空けました。これで準備完了！ サーバを立てたときにメモした「パブリック IP アドレス」を、ブラウザで開いてみました。するとしばらくぐるぐるした後で、「接続がタイムアウトしました」と表示されてしまいました。(図 3.26)

^{*8} Mac のターミナルはそもそも黒じゃなくて白ですね。



▲図 3.26 なぜか HTTP でサイトが表示されない…

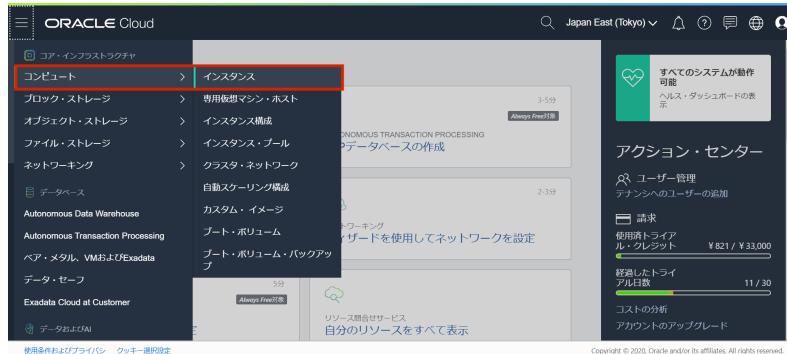
403 や 404 や、あるいは 502 などのステータスコードも返ってきていないので、そもそも NGINX までたどり着けていないようです。

3.7.1 サーバの手前にあるファイアウォールにも穴を空けよう

先ほどサーバの中にあるファイアウォールの設定を変更して、HTTP と HTTPS が通れるようにしましたが、実はサーバの中だけでなく、サーバの外にももう 1 つファイアウォールがいます。サイトが表示されなかったのは、「ウェブページを見せて！」というリクエストが、サーバの手前のファイアウォールで阻まれていたためなのです。サーバの手前にあるファイアウォールにも穴を空けて、HTTP と HTTPS がサーバまでたどり着けるようにしましょう。

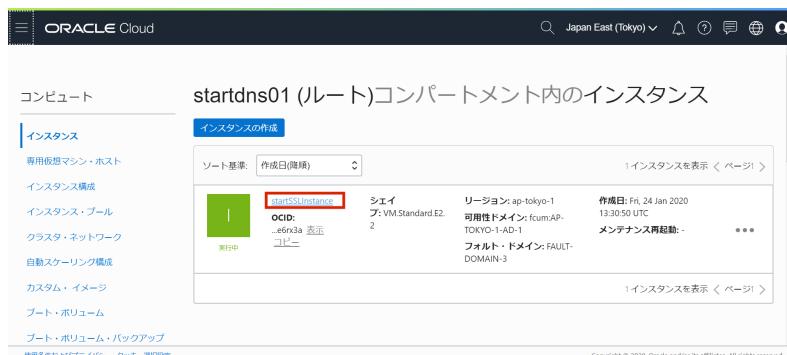
再び Oracle Cloud のコンソールに戻って、左上の [ハンバーガーメニュー] から [コンピュート] の [インスタンス] を開きます。(図 3.27)

3.7 なぜかサイトが見られない



▲図 3.27 [ハンバーガーメニュー] から [コンピュート] の [インスタンス] を開く

インスタンスの一覧が表示されるので [startSSLInstance] をクリックします。(図 3.28)



▲図 3.28 [startSSLInstance] をクリック

[パブリック・サブネット] をクリックします。(図 3.29)

第3章 ウェブサーバの設定をしよう

実行中

インスタンス情報

可用性ドメイン: iucm/AP-TOKYO-1-AD-1
フルト・ドメイン: FAULT-DOMAIN-3
リージョン: ap-tokyo-1
シティ: VM Standard E2.2
仮想クラウド・ネットワーク: VirtualCloudNetwork-20200120-2319
メンテナンス再起動: -

プライマリVNIC情報

プライベートIPアドレス: 10.0.0.2
パブリックIPアドレス: 140.238.33.51
ネットワーク・セキュリティ・グループ: None [編集]

内部FQDN: startdnsinstance... [表示] [コピー]
サブネット: [パブリック・サブネット] [Edit]

このインスタンスのトラフィックは、関連付けられた[サブネット](#)のセキュリティ・リストおよびVNICのネットワーク・セキュリティ・グループ

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

▲図 3.29 [パブリック・サブネット] をクリック

もう一度、[パブリック・サブネット] をクリックします。(図 3.30)

使用可能

CIDRブロック: 10.0.0.0/16
コンバートメント: startdns01 (ルート)
作成日: 2020年1月20日(月) 14:55:54 UTC

OCID: ..._httpa [表示] [コピー]
デフォルト・ルート: Default Route Table for VirtualCloudNetwork-20200120-2319
トポ:
DNSドメイン名: vcn.oraclecloud.com

リソース

サブネット(1)
ルート(1)
インターネット・ゲートウェイ(1)
動的ルーティング・ゲートウェイ(0)
ネットワーク・セキュリティ・グループ(0)

startdns01 (ルート) コンバートメント内のサブネット

サブネットの作成

名前	状態	CIDRブロック	サブネット・アクセス	作成日
[パブリック・サブネット]	● 使用可能	10.0.0.0/24	パブリック (リージョナル)	2020年1月20日(月) 14:55:56 UTC

Copyright © 2020, Oracle and/or its affiliates. All rights reserved.

▲図 3.30 もう一度、[パブリック・サブネット] をクリック

[セキュリティ・リスト] の中にある [Default Security List for VirtualCloudNetwork ~] をクリックします。(図 3.31) このセキュリティ・リストが、サーバの手前にいるファイアウォールです。

3.7 なぜかサイトが見られない

The screenshot shows the Oracle Cloud interface for a VirtualCloudNetwork named 'subnet... 東京 ネット'. It displays network details like CIDRブロック: 10.0.0.0/4, 仮想ルーターMACアドレス: 00:00:17:42:14:B4, andサブネット・タイプ: リージョナル. A link to the Default Route Table for VirtualCloudNetwork-20200120-2319 is shown. Below this, the 'セキュリティ・リスト' (Security List) section is visible, showing a single entry for 'Default Security List for VirtualCloudNetwork-20200120-2319' which is active.

▲図 3.31 [Default Security List for VirtualCloudNetwork～] をクリック

[イングレス・ルール] で、[イングレス・ルールの追加] をクリックします。(図 3.32)

The screenshot shows the 'Ingress Rule' creation screen. It lists three existing ingress rules and one pending rule. A red box highlights the 'Ingress Ruleの追加' (Add Ingress Rule) button. The table below shows two current rules: one allowing SSH traffic from 0.0.0.0/0 to port 22, and another allowing ICMP traffic from 0.0.0.0/0 to ports 3, 4.

ステータス	ソース	IPプロトコル	ソース・ポート範囲	宛先ポート範囲	タイプとコード	許可	説明
いいえ	0.0.0.0/0	TCP	All	22		いいえ	ポートのTCPトラフィック: 22 SSH Remote Login Protocol
いいえ	0.0.0.0/0	ICMP		3, 4		いいえ	次に対するICMPトラフィック: 3, 4宛先に到達できません: フラグメントデリケーションが必要ですが、フラグメント

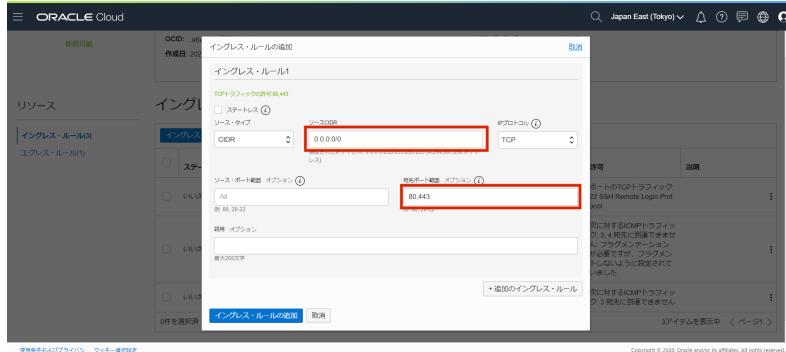
▲図 3.32 [イングレス・ルールの追加] をクリック

[ソース CIDR] に [0.0.0.0/0]^{*9}を入力します。[宛先ポート範囲] には [80,443]^{*10}を入力します。(図 3.33) どちらも入力できたら、[イングレス・ルールの追加] をクリックします。

^{*9} ソース CIDR は接続元の IP アドレス範囲のこと、0.0.0.0/0 はすべての IP アドレスを指します。つまり接続元がどんな IP アドレスでもファイアウォールを通れます、ということですね

^{*10} ポート番号とは、サーバという家や、その手前のファイアウォールという壁についているドアのようなものだと思ってください。同じサーバを訪問するときでも SSH は 22 番のドアを、HTTP は 80 番のドアを、HTTPS は 443 番のドアを通ります

第3章 ウェブサーバの設定をしよう



▲図 3.33 [ソース CIDR] に [0.0.0.0/0]、[宛先ポート範囲] には [80,443] を入力

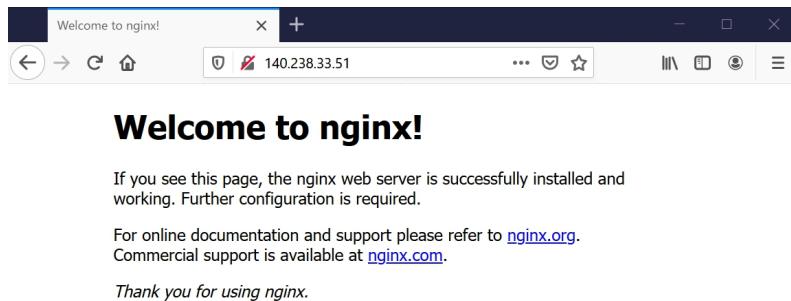
[イングレス・ルール] に、HTTP（80 番ポート）と HTTPS（443 番ポート）へのリクエストを通す設定が追加されました。（図 3.34）

ステータス	ソース	IPプロトコル	ソース・ポート範囲	宛先ポート範囲	タイプとコード	許可	説明
いいえ	0.0.0.0/0	TCP	All	22			ポートのTCPトラフィック 22 SSH Remote Login Protocol
いいえ	0.0.0.0/0	ICMP			3,4		次に対するICMPトラフィック 3,4宛先に到達できません ICMPメッセージのソースアドレス がマスクでマッチしないように設定さ れていました
いいえ	10.0.0.0/16	ICMP			3		次に対するICMPトラフィック 3宛先に到達できません
いいえ	0.0.0.0/0	TCP	All	80			ポートのTCPトラフィック 80
いいえ	0.0.0.0/0	TCP	All	443			ポートのTCPトラフィック 443 HTTPS

▲図 3.34 ルールが追加された！

3.7.2 今度こそ HTTP でサイトを見てみよう

再び、サーバを立てたときにメモした [パブリック IP アドレス] を、ブラウザで開いてみましょう。（図 3.35）

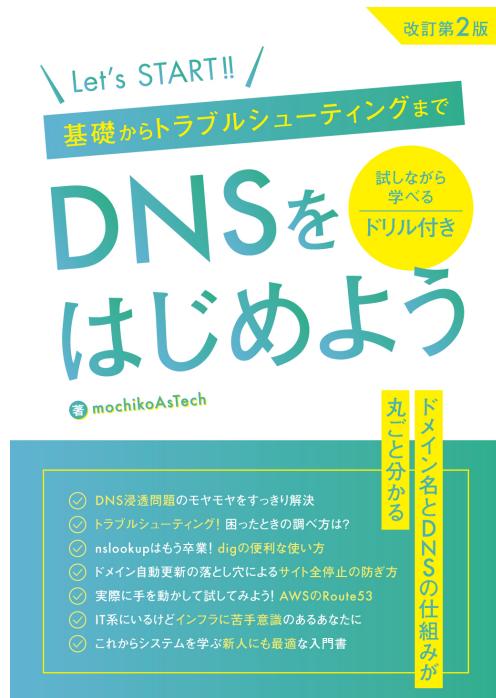


▲図 3.35 HTTP でサイトが見られた！

[Welcome to nginx!] と表示されました！ これでまず、「HTTP でサイトを表示する」はクリアです。

3.8 ドメイン名の設定をしよう

ウェブサーバの準備ができたので、HTTPS のサイト用にドメイン名を用意します。「自分のドメイン名？ そんなの持っていないよ！」という人は、先に「DNS をはじめよう」(図 3.36) で、ドメイン名を買ってからこの先へ進むようにしてください。



▲図 3.36 「DNS をはじめよう」(1,000 円) は BOOTH や Amazon (Kindle) で好評発売中

あなたが買ったドメイン名を使って「ssl. 自分のドメイン名」の A レコードを作成して、サーバの [パブリック IP アドレス] と紐付けてください。ネームサーバはお名前.comを使用してもいいですし、AWS の Route53 で設定しても構いません。

なお筆者が「DNS をはじめよう」で購入したのは startdns.fun というドメイン名だったので、ssl.startdns.fun という A レコードを作って、さっ立てたばかりのサーバの [パブリック IP アドレス] と紐付けます。例えばネームサーバが「お名前.com」なら、DNS 設定の画面でこのように A レコードを追加します。(図 3.37)

ホスト名	TYPE	TTL	VALUE	優先	状態	追加
ssl .startdns.fun	A ▾	3600	140 238 33 51		有効 ▾	追加

▲図 3.37 「ssl. 自分のドメイン名」の A レコードを作成する

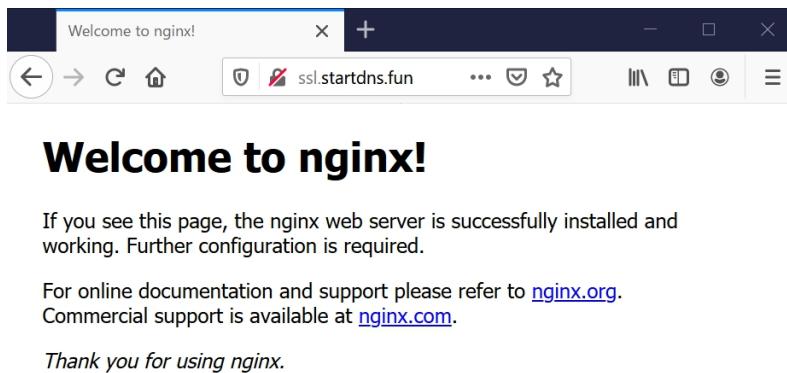
A レコードを追加できたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、サーバの IP アドレスが返ってくれれば A レコードは設定できています。

```
$ dig ssl. 自分のドメイン名 a +short
```

筆者の場合は、次のように表示されました。

```
$ dig ssl.startdns.fun a +short  
140.238.33.51
```

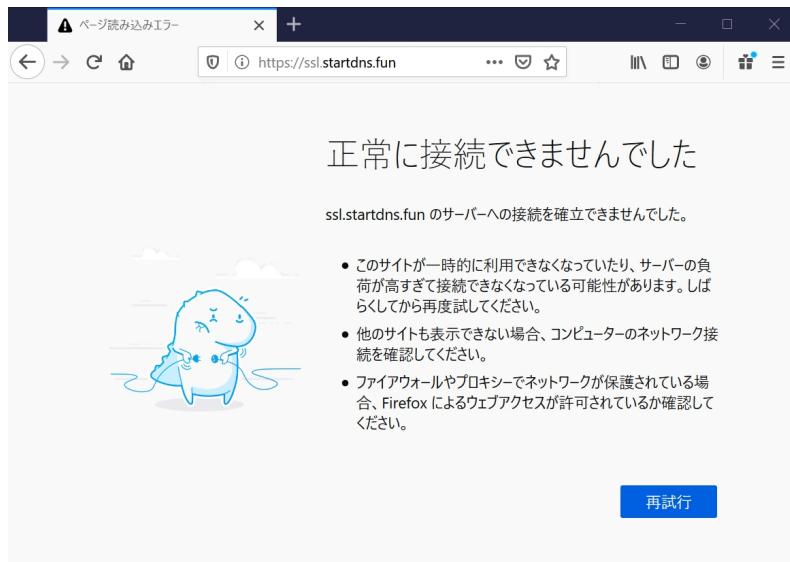
ドメイン名が設定できたら、ブラウザでも「http://ssl. 自分のドメイン名」を叩いてみましょう。先ほどと同じ NGINX のページが表示されるはずです。(図 3.38)



▲図 3.38 「ssl. 自分のドメイン名」でサイトが表示された！

HTTP でサイトを見ることができましたが、同じドメイン名を HTTPS で開いてみるとどうでしょう？ ブラウザで「https://ssl. 自分のドメイン名」を開いてみると、「正常に接続できませんでした」と表示（図 3.39）されました。

第3章 ウェブサーバの設定をしよう



▲図 3.39 HTTPS で開くと [正常に接続できませんでした] と表示された

それでは HTTPS でもサイトが見られるように、SSL 証明書を取得して設定をしていきましょう。

第4章

SSL 証明書を取得しよう

HTTP でサイトが見られたので、今度は HTTPS でも見られるよう、必要な材料を「SSL 証明書」を入手しましょう。

4.1 SSL 証明書にまつわる登場人物

SSL 証明書の取得は、関わってくる登場人物が多いので、最初に登場人物全員が集まる場所として、サーバの中で`/etc/nginx/`の下に`ssl`というディレクトリを作っておきます。^{*1}

```
# mkdir /etc/nginx/ssl/  
# cd /etc/nginx/ssl/
```

4.2 密密鍵（startssl.key）を作ろう

先ず最初に作成するのが密密鍵です。密密鍵は「.key」や「.pem」^{*2}という拡張子で作成されることが多いです。密密鍵は名前のとおり「密密」にすべきです。つまり、限られた人だけが触れるように管理すべきです。決してメールに添付して送ったり、サーバ内で誰でも見られる`/tmp/`以下に置いたりしてはいけません。

次の openssl コマンドを叩いて、密密鍵を生成しましょう。左から順に「openssl コマンドで、鍵アルゴリズムが RSA で、鍵の長さは 2048 ビットで、`/etc/nginx/ssl/`以下に`startssl.key`というファイル名で密密鍵を作って」という意味です。

```
# openssl genrsa 2048 >/etc/nginx/ssl/startssl.key  
Generating RSA private key, 2048 bit long modulus  
.....  
.....++  
e is 65537 (0x10001)
```

できあがった密密鍵を、`cat` コマンドで見てみましょう。

```
# cat /etc/nginx/ssl/startssl.key  
-----BEGIN RSA PRIVATE KEY-----  
MIIEpQIBAAKCAQEA0+s/Gzdb0bzg6QWzCvK5JofMv6izHzlCfMCMhcU7SeBd2tHN
```

^{*1} もし `mkdir: cannot create directory '/etc/nginx/ssl': Permission denied` と表示されてしまったら、あなたはいま、うっかり一般ユーザのままで `mkdir` コマンドを実行しています。コマンドの例で、左側のプロンプトが「#」のときは、root で実行してください。「`sudo su -`」と書いて Enter キーを押すと root になれます

^{*2} 密密鍵が PEM と呼ばれるテキスト形式で生成されることから、pem という拡張子が使われるようす

```
icRA7g5CZq09aaEqv1949cFX5C3bgHx140+epeudrKyUjRwZSpS70mznDBFQByTY  
(中略)  
InsCw9qu+iZknMKiISw3Krht/898/hq0jqLFJUTbfg9BP8w+JVW4+8hp40Sklymc  
NRcvPYUBQy3wK+w527rksodkGZ77c6Q+XxRtH/wpo3H+xwhmJvi+T2o=  
-----END RSA PRIVATE KEY-----
```

【コラム】SSL証明書の秘密鍵にパスフレーズは設定すべき？

openssl コマンドで秘密鍵を作るとき、-aes128 や-aes256 というオプションを付けると、パスフレーズを聞かれて、指定の暗号で暗号化された秘密鍵が出来上がります。^{*3} [Enter pass phrase:] や [Verifying - Enter pass phrase:] と表示された際に、いくらかキーボードを叩いて入力しても、黒い画面上は何も表示されません。ですが、ちゃんと文字入力はできているので大丈夫です。入力を間違えたら Backspace キーで消して、書き直すこともできます。

```
# openssl genrsa -aes128 2048 >/etc/nginx/ssl/with-passphrase.key  
Generating RSA private key, 2048 bit long modulus  
(中略)  
Enter pass phrase: ←秘密鍵のパスフレーズとして「startssl」と入力  
Verifying - Enter pass phrase: ←もう一度「startssl」と入力
```

秘密鍵がパスフレーズで保護されていると、秘密鍵を盗んだ誰かが使おうとしても、パスフレーズが分からなければ使えないで安心です。

しかし秘密鍵にパスフレーズが設定されていると、Apache や Nginx といったウェブサーバを再起動した際にも、必ずパスフレーズを聞かれます。もし何かトラブルがあってサーバが OS ごと再起動してしまった場合、折角 Nginx が自動起動する設定になっていても、パスフレーズを聞くところで止まって起動できず、サイトが自動復旧しない、というトラブルが発生します。

これを回避するには、パスフレーズをファイルに書いておいて、Nginx の自動起動時にそれを読み込むようにする、という方法があります。しかし折角設定したパスフレーズをファイルに書いて同じウェブサーバ内に置いてしまうと、秘密鍵を盗むとき一緒にパスフレーズのファイルも盗まれてしまう可能性が高くなり、もはやパスフレーズを設定した意味がありません。そのため筆者は、ウェブサーバに設置して使う秘密鍵にはパスフレーズは設定しなくてよい、という考えです。

ちなみに、次のように一度パスフレーズありで作った秘密鍵を、パスフレーズなしで複製することも可能です。

```
# cd /etc/nginx/ssl/  
# openssl rsa -in with-passphrase.key -out without-passphrase.key  
Enter pass phrase for with-passphrase.key: ← 秘密鍵のパスフレーズ  
[startssl] を入力  
writing RSA key
```

利便性を保つつつ安全性も向上させたければ、本番のウェブサーバで使う秘密鍵はパスフレーズなし、バックアップサーバで保管しておく秘密鍵はパスフレーズありにしておく、という方法がいいでしょう。

```
# cd /etc/nginx/ssl/  
# openssl req -new -key startssl.key -out startssl.csr
```

CSRを作成するため、いくつか質問をされます。(表4.1)^{*4}

^{*3} さっきは-aes128 や-aes256 といった「どの暗号で暗号化するか？」のオプションを何も指定しなかったので、パスフレーズは聞かれず、秘密鍵も暗号化されませんでした

^{*4} 本著では DV 証明書を取得するため「組織単位名 (OU)」は任意としていますが、証明書の種類や認証局によっては必須のところもあるようです。取得時に認証局のサイトで確認しましょう。DV 証明書については後述します

4.3 CSR (`startssl.csr`) を作ろう

▼表 4.1 CSR 作成時に聞かれる質問

必須/任意	質問	説明	入力する内容
必須	Country Name	国名 (C)	JP
必須	State or Province Name	都道府県名 (S/ST)	Tokyo
必須	Locality Name	市町村名 (L)	Shinjuku-ku
必須	Organization Name	組織名 (O)	mochikoAsTech
任意	Organizational Unit Name	組織単位名 (OU)	入力なし
必須	Common Name	コモンネーム (CN)	ssl.startdns.fun

```
Country Name (2 letter code) [XX]:JP
State or Province Name (full name) []:Tokyo
Locality Name (eg, city) [Default City]:Shinjuku-ku
Organization Name (eg, company) [Default Company Ltd]:mochikoAsTech
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:ssl.startdns.fun
```

次の 3 つは入力不要です。

```
Email Address []:
A challenge password []:
An optional company name []:
```

質問に全て答えると、CSR ができあがります。「Subject」と書かれた行を見て、生成された CSR の内容が正しいか確認しましょう。

```
# openssl req -text -in /etc/nginx/ssl/startssl.csr -noout | head -4
Certificate Request:
Data:
Version: 0 (0x0)
Subject: C=JP, ST=Tokyo, L=Shinjuku-ku, O=mochikoAsTech, CN=ssl.startdns.fun
```

CSR を cat コマンドで表示して、「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをメモしておきましょう。メモした CSR はこの後使用します。

```
# cat /etc/nginx/ssl/startssl.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICqzCCAZMCAQAwZjELMAkGA1UEBhMCS1AxDjAMBgNVBAgMBVRva3lvMRQwEgYD
```

```
VQQHDAtTaGluanVrdS1rdTEWMBQGA1UECgwNbW9jaGlrbOFzVGVjaDEZMBcGA1UE  
(中略)  
jP1RMQS3PuYDE6QIVJ5zbMC+RIydSQ/ODr9VUHWiYqDPjx+BpphYT5AxMwbw9/m5  
noKGVJ11Mt7G03Awa/TX  
-----END CERTIFICATE REQUEST-----
```

4.3.1 【ドリル】CSRで入力すべきなのはクライアントの情報？

問題

A銀行のウェブサイトをHTTPSで作ることになりました。SSL証明書^{*5}の取得はA銀行の代わりに広告代理店のB社が行い、さらにサイトの制作や運用はA銀行からWeb制作会社のC社に委託する場合、CSRで入力する住所や会社名はA銀行・B社・C社のどれにすべきでしょうか？

- A. A銀行のウェブサイトなんだからA銀行を入力すべき
- B. A銀行から任されてSSL証明書を買うのはB社だからB社を入力すべき
- C. 実際にサイトの管理を任せているのはC社だからC社を入力すべき

答え _____

解答

正解はAです。ウェブサイトの運営元がA銀行であることを証明するためのSSL証明書なので、CSRではA銀行（クライアント）の情報を記載すべきです。

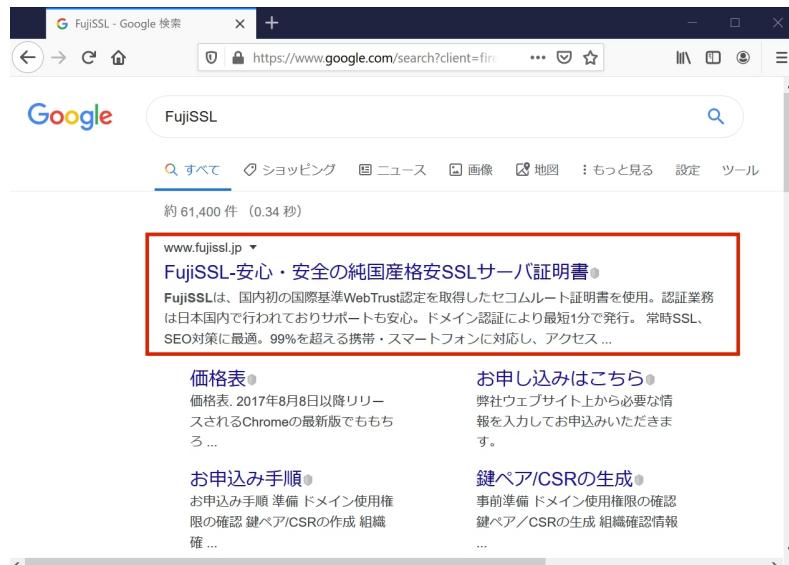
A銀行のフィッシングサイトが出てきたときに、エンドユーザが「本物のサイトか確認しよう」と思って証明書の情報を見て、B社やC社の情報が表示されたら「A銀行じゃない！」となってしまいます。

4.4 SSL 証明書の取得申請を出そう

[FujiSSL]で検索して、[FujiSSL-安心・安全の純国産格安SSLサーバ証明書]をクリック（図4.1）します。

^{*5} SSL証明書の種類は「EV証明書」とします。EV証明書については後述します

4.4 SSL 証明書の取得申請を出そう



▲図 4.1 [Fujissl-安心・安全の純国産格安SSLサーバ証明書] をクリック

右上の「お申し込みはこちら」をクリック（図 4.2）します。



▲図 4.2 [お申し込みはこちら] をクリック

続いて「お申し込みサイト（ストアフロント）へ」をクリック（図 4.3）します。

第4章 SSL 証明書を取得しよう



▲図 4.3 [お申し込みサイト（ストアフロント）へ] をクリック

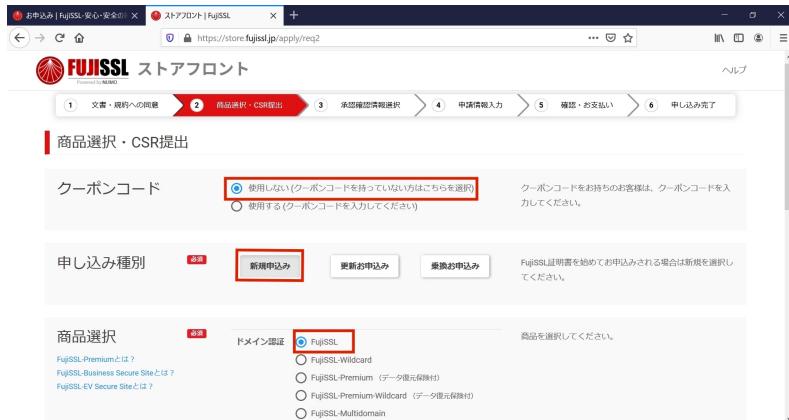
指定された文書と「収集した個人情報の利用目的」を確認した上で、チェックボックスにチェックを入れて [次へ] をクリック（図 4.4）します。



▲図 4.4 チェックを入れて [次へ] をクリック

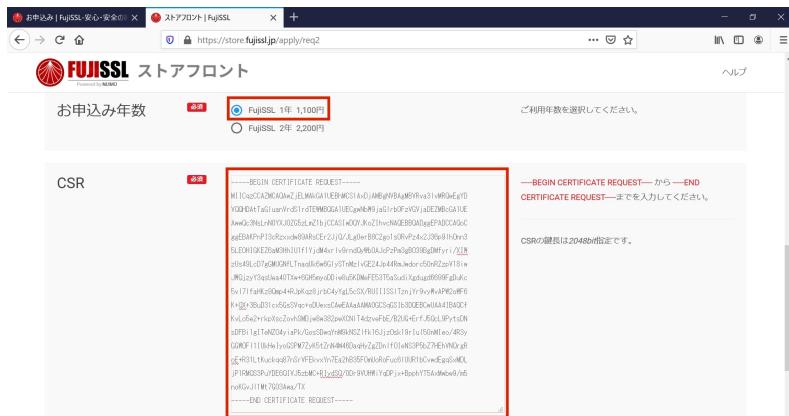
クーポンコードは「使用しない」、申し込み種別は「新規申し込み」、商品選択は「ドメイン認証」の「FujiSSL」になっていることを確認（図4.5）します。

4.4 SSL 証明書の取得申請を出そう



▲図 4.5 クーポンコード、申し込み種別、商品選択を確認

下へスクロールして、お申込み年数が「[FujiSSL 1年 1,100円]」^{*6}になっていることを確認（図 4.6）したら、CSR に、先ほど「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをメモしておいた CSR をペーストします。

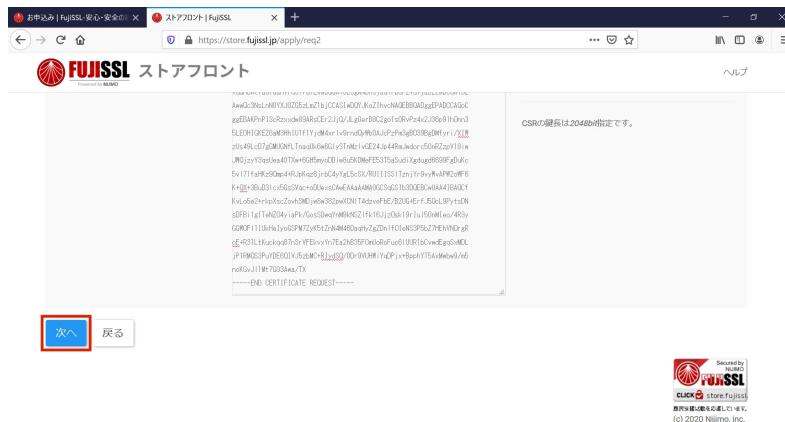


▲図 4.6 CSR をペーストする

[次へ] をクリック（図 4.7）してください。

^{*6} 2019年2月現在、FujiSSL の「ドメイン認証シングルタイプ」というSSL証明書は、有効期間1年で税込1,100円です

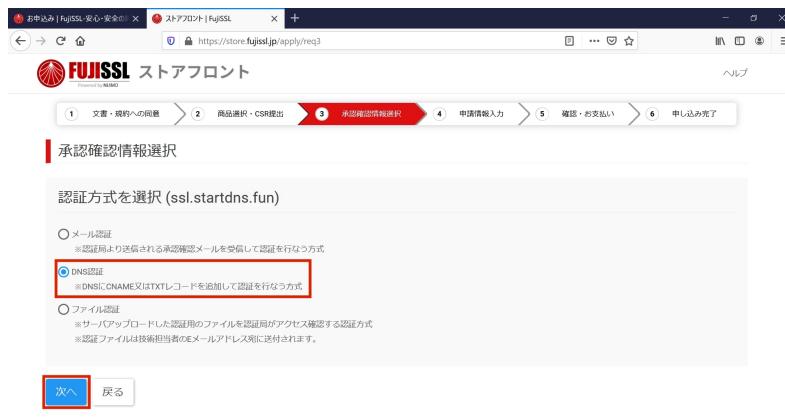
第4章 SSL 証明書を取得しよう



▲図 4.7 [次へ] をクリック

認証方式は「DNS 認証」を選択（図 4.8）してください。これは「CSR のコモンネームで指定したドメイン名が、あなたの持ち物であることをどうやって証明しますか？」ということを聞かれています。対象のドメイン名で、メールを受信して URL を踏むか、指定された内容のリソースレコードを DNS で追加するか、サイトに指定された内容のファイルをアップするか、いずれかの方法で「このドメイン名（筆者なら ssl.startdns.fun）は私の持ち物です」ということを証明しなければいけません。

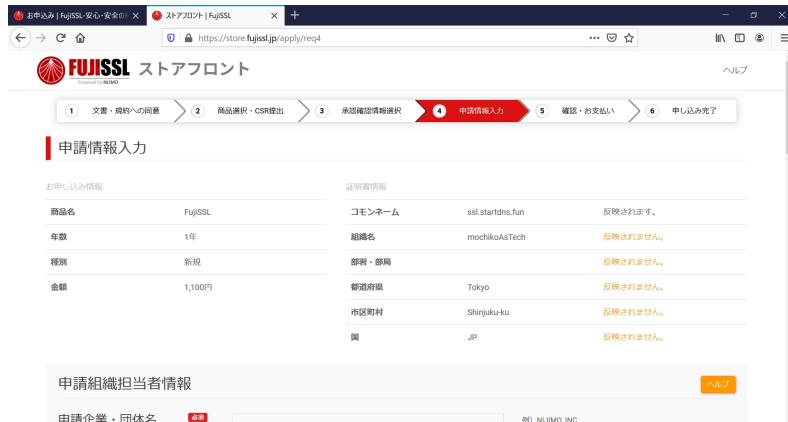
今回は DNS で TXT レコードを追加する、という方法で証明するので、「DNS 認証」を選択して「次へ」をクリックします。



▲図 4.8 [DNS 認証] を選択して [次へ] をクリック

4.4 SSL 証明書の取得申請を出そう

今回購入するのは DV 証明書です。DV 証明書は「そのドメイン名の使用権があること」だけを証明してくれます。サイトの運営者が日本にいることや、東京の新宿区にオフィスがあること、mochikoAsTech という組織であることなどは確認も証明もしないので、実際の証明書にもそれらの情報は反映されません。（図 4.9）



▲図 4.9 [申請情報入力] 画面

[申請組織担当者情報] と [技術担当者情報] を英語表記で入力（図 4.10）します。ここで入力した住所や電話番号、個人名などは外向けに公開されることはありませんので、安心して入力してください。^{*7}この後、ここで入力した [E メールアドレス] 宛てに、SSL 証明書が送られてきます。メールアドレスを間違えないよう注意してください。すべて入力したら、[次へ] をクリック（図 4.11）します。

^{*7} EV 証明書や OV 証明書の場合は、ここで入力した担当者宛てに実在確認の連絡がります。詳細については後述します

第4章 SSL 証明書を取得しよう

The screenshot shows the FujSSL application interface. The main title is 'FujSSL ストアフロント'. The left sidebar has a tree view with 'お申込み' (Application), '支払い方法' (Payment Method), 'SSL証明書' (SSL Certificate), and 'お問い合わせ' (Contact). The right sidebar has tabs for 'ヘルプ' (Help), 'FAQ', and 'お問い合わせ' (Contact). The main content area is divided into two sections:

- 申請組織担当者情報 (Organization Contact Information):**
 - 申請企業・団体名: mochikoAsTech (例) NIMO, INC. (※個人の場合は個人名を入力してください)
 - お名前: 姓: Nekomura, 名: Mochiko (例) 姓: Fuji, 名: Taro
 - 役職: Technical Writer (例) CEO
 - 国名: 日本 (Japan)
 - 郵便番号: 157-0071 (例) 000-0000
 - 所在地: Tokyo (例) Tokyo
 - 詳細: Shinjuku-ku (例) Shibuya-ku
 - 詳細: 4-1-6 Shinjuku (例) 1-12-2, Shibuya
 - 詳細: SHINJUKU MIRAINA TOWER (例) CROSS OFFICE SHIBUYA 5F
- 技術担当者情報 (Technical Contact Information):**
 - 所在地: Tokyo
 - 詳細: Shinjuku-ku
 - 詳細: 4-1-6 Shinjuku
 - 詳細: SHINJUKU MIRAINA TOWER
 - 電話番号: 03-XXXX-XXXX (例) 03-0000-0000
 - Eメールアドレス: startdns.01@gmail.com (例) taro@fujssl.jp (証明書送付先メールアドレスになります)

▲図 4.10 [申請組織担当者情報] と [技術担当者情報] を入力

The screenshot shows the FujSSL application interface. The main title is 'FujSSL ストアフロント'. The left sidebar has a tree view with 'お申込み' (Application), '支払い方法' (Payment Method), 'SSL証明書' (SSL Certificate), and 'お問い合わせ' (Contact). The right sidebar has tabs for 'ヘルプ' (Help), 'FAQ', and 'お問い合わせ' (Contact). The main content area shows the same contact information as in Figure 4.10, but the '次へ' (Next) button at the bottom is highlighted.

▲図 4.11 [次へ] をクリック

[決済情報入力] に、SSL 証明書代を支払うクレジットカード情報を入力（図 4.12）します。

4.4 SSL 証明書の取得申請を出そう

The screenshot shows a web browser window for 'FujSSL Storefront'. The main title is '決済情報入力' (Payment Information Input). It displays a payment amount of '1,100 円' with a note '(※税込み)'. Below this, there's a section for 'ご利用いただけるカードの種類' (Card Types Available) with icons for American Express, MasterCard, VISA, JCB, Diners Club, and UnionPay. The 'クレジットカード番号' (Credit Card Number) field is filled with a redacted number and includes a note about expiration dates. The '有効期限' (Expiration Date) field shows a dropdown menu for month and year. The 'セキュリティコード' (Security Code) field is also redacted. The 'カード名義' (Cardholder Name) field contains 'TARO FUJI'. At the bottom, there's a section for '書類送付先' (Document Delivery Address) which is currently empty.

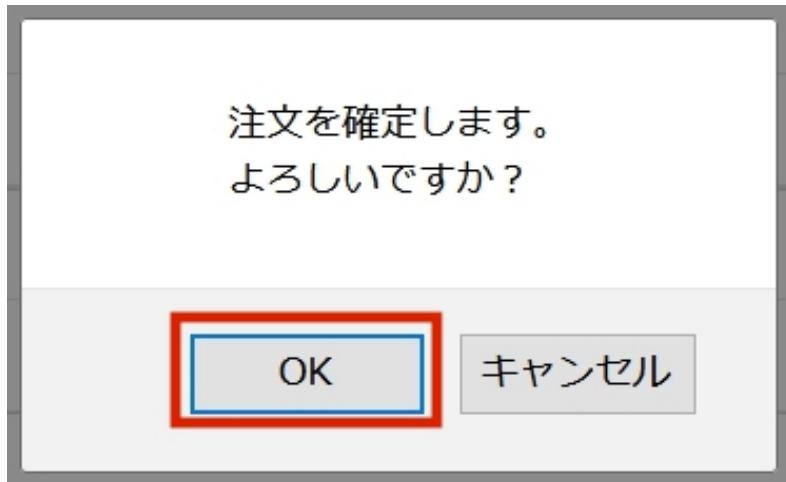
▲図 4.12 クレジットカード情報を入力

【書類送付先】で【請求書宛名】、【納品書宛名】、【領収書宛名】を記入したら、【上記の内容で注文を確定する】をクリック（図 4.13）します。

The screenshot shows the same 'FujSSL Storefront' interface. This time, the '書類送付先' (Document Delivery Address) section is populated with three entries, each with a red border around the input field. The first entry is '請求書宛名' (Bill of Lading Recipient) with value 'mochikoAsTech'. The second is '納品書宛名' (Delivery Note Recipient) with value 'mochikoAsTech'. The third is '領収書宛名' (Receipt Recipient) with value 'mochikoAsTech'. Below these fields is a button labeled '上記の内容で注文を確定する' (Confirm Order with Above Content). At the bottom right, there's a security seal from 'CLICK & STORE FujSSL'.

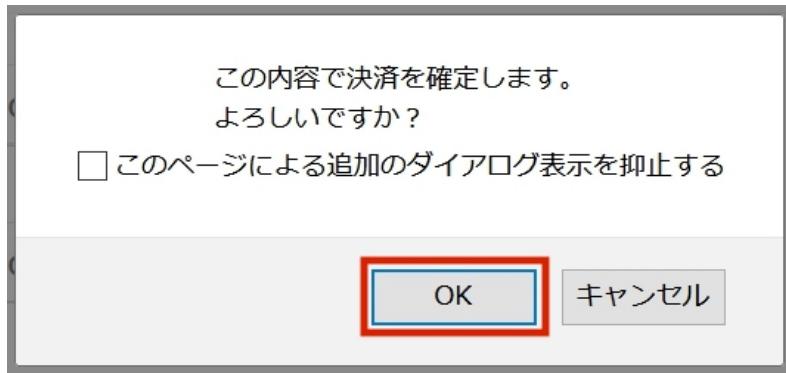
▲図 4.13 【上記の内容で注文を確定する】をクリック

「注文を確定します。よろしいですか？」と表示（図 4.14）されたら、[OK] をクリックします。



▲図 4.14 [OK] をクリック

「この内容で決済を確定します。よろしいですか？」と表示（図 4.15）されます。「1年間有効な SSL 証明書を 1,100 円で買うんだ！」というケツイ^{*8}をしたら、[OK] をクリックします。



▲図 4.15 1,100 円払うケツイをして [OK] をクリック

お申し込み完了のページが表示（図 4.16）されました。先ほど登録したメールアドレス宛に、DNS 設定情報を知らせるメールが届いていますので確認しましょう。

^{*8} UNDERTALE というゲームでは「ケツイを ちからに かえるんだ…！」という台詞が繰り返し出でています

4.5 DNS の設定をしよう

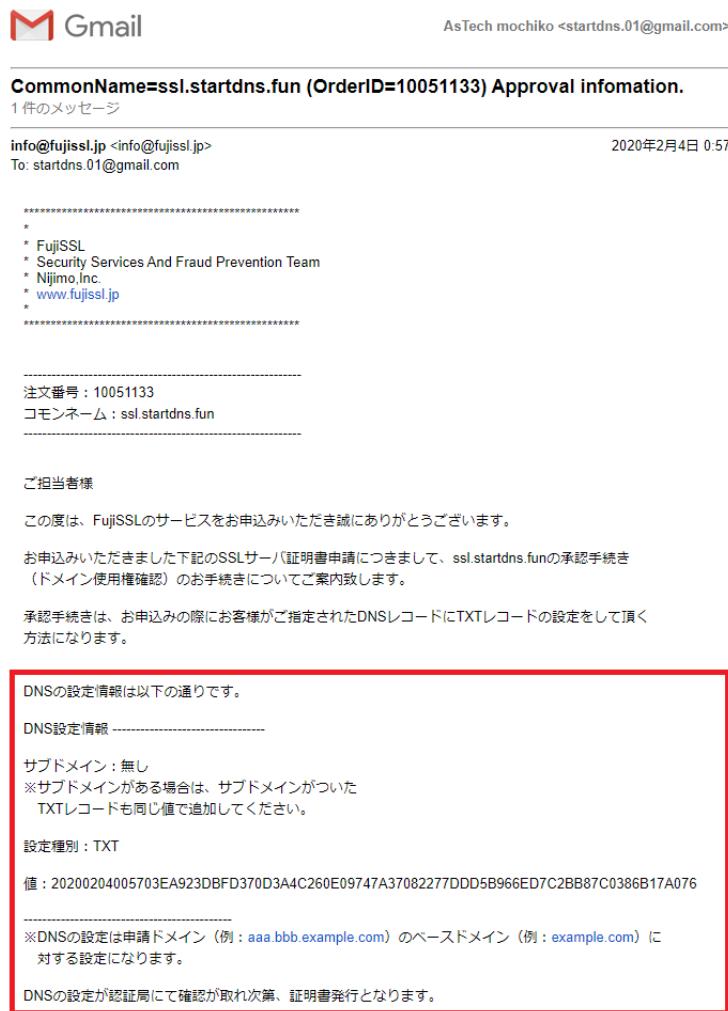


▲図 4.16 [お申込み完了] 画面

4.5 DNS の設定をしよう

[CommonName=ssl. 自分のドメイン名] から始まる件名のメール（図 4.17）がすぐに届きました。「CSR のコモンネームで指定したドメイン名が ssl.startdns.fun の場合、startdns.fun の TXT レコードを追加して、メールに書いてある値を設定するよう書いてあります。

第4章 SSL証明書を取得しよう



▲図 4.17 追加すべき TXT レコードの値がメールで届いた

例えばネームサーバが「お名前.com」なら、DNS 設定の画面でこのように TXT レコードを追加（図 4.18）します。今回はサブドメインを含まないドメイン名を追加するので、[ホスト名] には何も入力しません。[VALUE] には、メールに書いてあった値をそのままコピペーストします。

4.5 DNS の設定をしよう

A/AAAA/CNAME/MX/NS/TXT/SRV/DS/CAAレコード						
ホスト名	TYPE	TTL	VALUE	優先	状態	追加
.startdns.fun	TXT ▾	3600	20200204005703EA923DBF		有効 ▾	追加

▲図 4.18 「自分のドメイン名」の TXT レコードを追加する

TXT レコードができたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、サーバの IP アドレスが返ってくれば A レコードは設定できています。

```
$ dig 自分のドメイン名 txt +short
```

筆者の場合は、次のように表示されました。

```
$ dig startdns.fun txt +short  
"20200204005703EA923DBFD370D3A4C260E09747A37082277DDD5B966ED7C2BB87C0386B17A076"
```

TXT レコードを追加してから、おおよそ 30 分後に SSL 証明書がメール（図 4.19）で届きました。

Gmail AsTech mochiko <startdns.01@gmail.com>

(OrderID=10051133) FujiSSL Fulfilment E-mail

2件のメッセージ

info@fujissl.jp <info@fujissl.jp> 2020年2月4日 1:30
To: startdns.01@gmail.com

*
* FujiSSL
* Security Services And Fraud Prevention Team
* Nijimo,Inc.
* www.fujissl.jp
*

注文番号 : 10051133
コモンネーム : ssl.startdns.fun

ご担当者様
このたびはFujiSSLのサービスをご利用いただき誠にありがとうございます。
お客様の証明書が発行いたしましたのでご案内いたします。
証明書はこのメールの下部に貼付されております。
当メールの添付ファイルから証明書一式を取得いただくことも可能です。

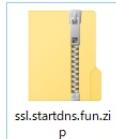
▲図 4.19 SSL 証明書がメールで届いた

メールに添付されている ZIP ファイル (ssl_自分のドメイン名.zip) に SSL 証明書が入っていますのでダウンロードします。(図 4.20)



▲図 4.20 メールに添付されている ZIP ファイル

Windows の方も Mac の方も、ダウンロードした ZIP ファイルはデスクトップに置いてください。(図 4.21)



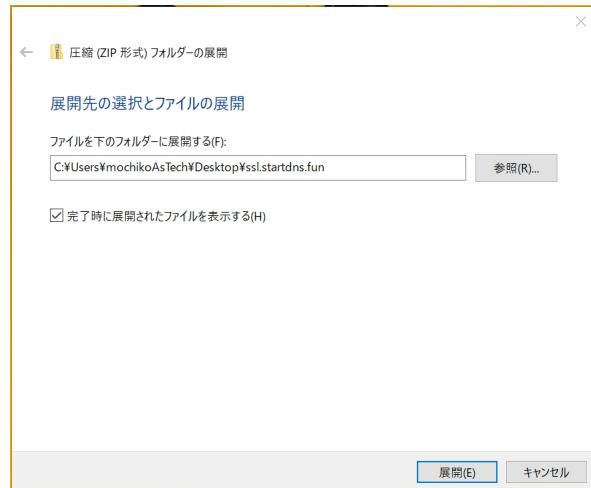
▲図 4.21 ダウンロードした ZIP ファイルはデスクトップに置く

ZIP ファイルを右クリックして、[すべて展開] をクリックします。(図 4.22)



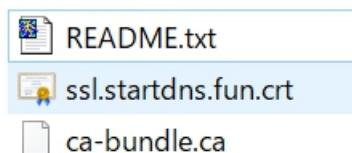
▲図 4.22 右クリックして [すべて展開] をクリック

[展開] をクリックします。(図 4.23)



▲図4.23 [展開] をクリック

展開したフォルダ（図4.24）の中の「server.crt」がSSL証明書で、「ca-bundle.ca」が中間CA証明書です。READMEはファイルの説明書です。



▲図4.24 server.crt が SSL 証明書で、ca-bundle.ca が中間 CA 証明書

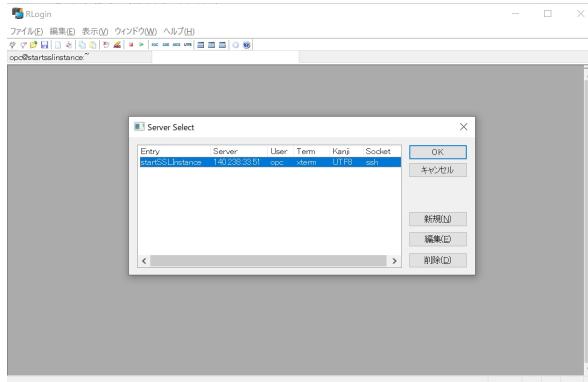
どちらも必要なものなので、この2つのファイルをサーバにアップロードしましょう。

4.6 SSL証明書をサーバに設置しよう

それではSSL証明書をサーバに設置します。NGINXではSSL証明書と中間CA証明書の2つを、1ファイルにまとめて使用するので、まずはアップロードして、それから1ファイルにまとめる作業を行ないます。

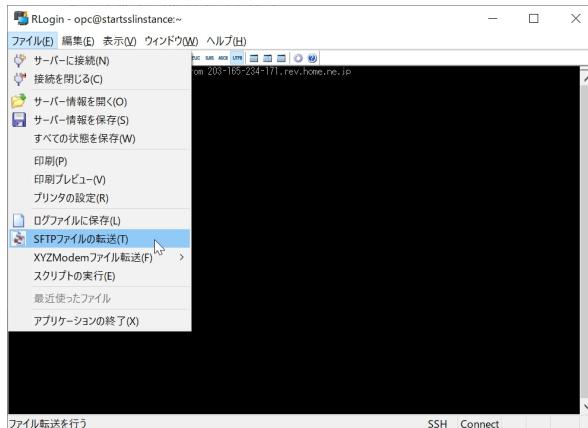
4.6.1 Windows で証明書と中間 CA 証明書をアップしよう

Windows のパソコンを使っている方は、RLogin を起動します。[startSSLInstance] を選択して、[OK] をクリック（図 4.25）します。



▲図 4.25 RLogin を起動して [startSSLInstance] を選択してログイン

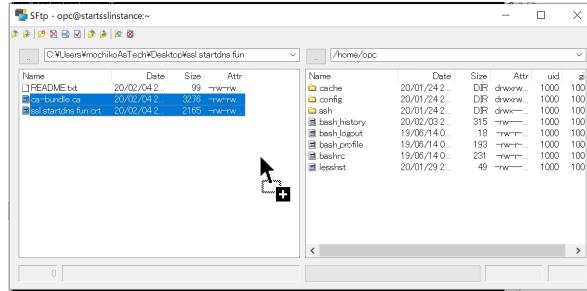
黒い画面が開いたら、[ファイル] から [SFTP ファイルの転送] をクリック（図 4.26）します。



▲図 4.26 [ファイル] から [SFTP ファイルの転送] をクリック

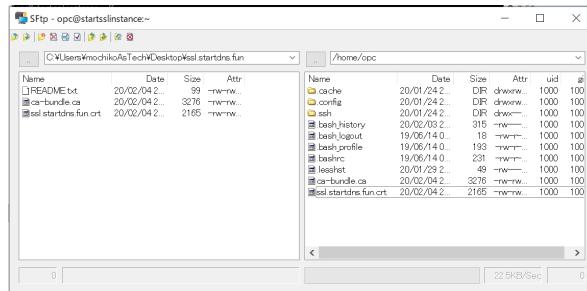
左側があなたのパソコンで、右側がサーバです。左側をデスクトップに展開したフォル

ダ^{*9}にしたら、[server.crt] と [ca-bundle.ca] を右側にドラッグ&ドロップ（図 4.27）してください。これでサーバの「/home/opc」にファイルがアップロードされます。



▲図 4.27 [ファイル] から [SFTP ファイルの転送] をクリック

右側に [server.crt] と [ca-bundle.ca] がアップされたら、×を押してファイル転送の画面は閉じて構いません。（図 4.28）



▲図 4.28 [ファイル] から [SFTP ファイルの転送] をクリック

4.6.2 Mac で証明書と中間 CA 証明書をアップしよう

Mac を使っている方は、ターミナルを起動してください。scp コマンドを使って、Mac の中にある [server.crt] と [ca-bundle.ca] を、サーバにアップロードします。[展開したフォルダ] と [パブリック IP アドレス] の部分は、ご自身のものに書き換えてください。

*9 筆者の場合は「C:\Users\mochikoAsTech\Desktop\ssl.startdns.fun」でした

```
$ cd ~/Desktop/展開したフォルダ
$ scp -i ~/Desktop/startSSLKey server.crt ca-bundle.ca opc@パブリック IP アドレス:/home/opc/
server.crt 100% 0 0.0KB/s 00:00
ca-bundle.ca 100% 0 0.0KB/s 00:00
```

4.6.3 SSL 証明書と中間 CA 証明書を 1 ファイルにまとめよう

サーバにログインしている状態で、次のコマンドを叩いて、先ほどアップロードした [server.crt] と [ca-bundle.ca] が、ちゃんと「/home/opc/」以下に存在していることを確認します。

```
$ sudo su -
# ls /home/opc/
ca-bundle.ca  ssl.startdns.fun.crt
```

続いて 2 つのファイルから、新たに [startssl.crt] というファイルを作りましょう。^{*10}NGINX では SSL 証明書と中間 CA 証明書という 2 つのファイルを、1 つのファイルにがっちゃんとつなげて使うためです。

```
# cd /etc/nginx/ssl/
# awk 1 /home/opc/ssl.startdns.fun.crt /home/opc/ca-bundle.ca > startssl.crt
```

cat コマンドで [startssl.crt] を確認してみましょう。次のように「----BEGIN CERTIFICATE----」と「----END CERTIFICATE----」が、繰り返し 3 つ表示されれば大丈夫です。

```
# cat /etc/nginx/ssl/startssl.crt
-----BEGIN CERTIFICATE-----
MIIF/DCCBOSgAwIBAgIQaoS/P1b4mCR8mn5/WUrI6zANBgkqhkiG9wOBAQsFADBn
MQswCQYDVQQGEwJKUDE1MCMGA1UEChMcU0VDT00gVHJ1c3QgU3lzdGVtcyBDTy4s
(中略)
31pTIUPabkFxDPjs1Vw9c7z3Vgk2fnpuwE2lrE+46zrJ3oTRqsABDbYreK1a5vsG
tcnpUll1hrk/rC3JuI2ttHVaHU+1JCgTSRpv03a44azDy15T5C97dGxuowPgcaMQ
-----END CERTIFICATE-----
```

^{*10} このとき cat コマンドで結合すると、SSL 証明書と中間 CA 証明書の間に改行が入らず、「----END CERTIFICATE-----BEGIN CERTIFICATE----」のようになってしまふため、awk コマンドを使っています

```
-----BEGIN CERTIFICATE-----
MIIEpzCCA4+gAwIBAgIJIrnxVPM8X14AMAOGCSqGSIb3DQEBCwUAMF0xCzAJBgNV
BAYTAkpQMSUwIwYDVQQKExxTRUNPTSBUcnVzdCBTeXNOZW1zIENPLixMVEQuMScw
(略)
g3tiJAlFIpfXXD4cArZo6Z1XJ26B4H7vk5GmyR6poDy/CRvC7VIz3xp6o2348W1j
32S9pEuZhxtMvPjnsHIWPNdz8pHv21x7bYwDnocwN2uk3Qrr1jxTQ9evg==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIEcjCCA1qgAwIBAgIJERmw+nLg2EjGMAOGCSqGSIb3DQEBCwUAMFAxCzAJBgNV
BAYTAkpQMRgwFgYDVQQKEw9TRUNPTSBUcnVzdC5uZXQxJzA1BgNVBAstH1N1Y3Vy
(略)
u5ZuCjxerxj3qS1rM46bcEfjopnaD7hnJXSYiL1d0yw5zSW2PEe+LHdoIAb2I6D8
8UFJHOCLi6sY5l8jhjk0Os1yeu1C/RcY0+NBNHKZkFEeEb6ezOsg=
-----END CERTIFICATE-----
```

これでファイルの準備は完了です。

4.7 NGINX で HTTPS のバーチャルホストを作ろう

「/etc/nginx/conf.d/」というディレクトリの下で、もともとあった設定ファイルをバックアップしておきます。

```
# cd /etc/nginx/conf.d/
# mv default.conf default.conf.backup
```

vi コマンドで、同じ場所に新しい設定ファイルを作ります。vi コマンドでファイルを編集するときは、i（アイ）を押してから入力します。書き終わったら ESC キーを押して、「:wq」と入力して Enter キーを押せば変更が保存されます。

```
# vi startssl.conf
server {
    listen 80 default_server;
    return 301 https://$host$request_uri;
}

server {
    listen      443 ssl http2;
    server_name ssl.startdns.fun;

    # 密密鍵
    ssl_certificate_key /etc/nginx/ssl/startssl.key;
    # SSL 証明書
    ssl_certificate     /etc/nginx/ssl/startssl.crt;

    # 暗号スイート
    ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-A
    # プロトコルバージョン
```

```
ssl_protocols      TLSv1.2;
# 暗号スイートの順序はサーバが決める
ssl_prefer_server_ciphers  on;

location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
}
}
```

設定ファイルが書けたら、構文エラーがないかテストをします。もし書き間違いがあれば、ここでエラーメッセージとして表示されます。

```
# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

[test is successful] と表示されたら、NGINX を再起動して設定を反映しましょう。

```
# systemctl restart nginx.service
```

それからアクセスしたときに表示される、index.html の文字も [Let's start SSL/TLS] に変えておきましょう。

```
# cd /usr/share/nginx/html/
# cp -p index.html index.html.backup
# echo "Let's start SSL/TLS" > index.html
```

4.8 HTTPS でサイトを開いてみよう

証明書を設置して、NGINX の設定ファイルを修正し、NGINX の再起動もしたのでブラウザで「http://ssl.自分のドメイン名」を開いてみましょう。鍵マーク付きで HTTPS のページが表示されるはずです。(図 4.29)



▲図 4.29 HTTPS でサイトが表示された！

【コラム】ロードバランサーで SSL ターミネーションする方法もある

なお本著ではウェブサーバに SSL 証明書を設置しましたが、ウェブサーバの手前にロードバランサを置いて、そこに SSL 証明書を設置して、SSL ターミネーションを行う方法もあります。その場合、SSL で通信するのはエンドユーザのパソコンから終端となるロードバランサーまでで、ロードバランサーとウェブサーバの間は HTTP で通信するのが一般的です。

この方法には、次のようなメリットがあります。

- 暗号化や復号の処理を行う終端がロードバランサになるので、ウェブサーバの処理負荷が下がる- (AWS の ELB と ACM の場合は) 証明書の取得や更新、設置が自動で行われる

第 5 章

SSL/TLS について学ぼう

この章では SSL/TLS について学びます。

5.1 「サイトを HTTPS 化する」とは何か？

そもそもですが「サイトを HTTPS 化する」とは、なんでしょう？

サイトの HTTPS 化とは、サイト全体を「https://」から始まる URL にすることで、「常時 SSL 化」や「常時 SSL/TLS 化」あるいは「AOSSL (Always On SSL の略)」などとも呼ばれています。

数年前まではお問い合わせや会員登録など、個人情報を入力したり表示したりする一部のページのみ HTTPS にしているサイトが多かったですが、これをウェブサイト全体に適用しましょう、というのがサイトの HTTPS 化です。

5.2 どんなサイトでも必ず HTTPS にしなきゃだめ？

でも企業がやっているネットショップならまだしも、個人情報をやり取りするわけでもない、ただ日記を公開しているだけといった個人のサイトも HTTPS 化しなければいけないのでしょうか？ サイトを HTTPS 化するメリットには、

- 検索順位が上がる- サイトの表示が速くなる- アクセス解析の精度があがる

などがよく挙げられます。確かに上記のようなメリットはありますが、2014年より前は「やらないと何かまずいことが起こる」というわけではありませんでした。

そのため「やらなければ…」と思いつつ、先送りにしてきたケースも多いと思います。ですが2020年時点では、HTTPS化しないでいるとさまざまなデメリットが発生します。どんなデメリットが起きるのか、具体的に解説していきましょう。

5.3 HTTPS のままだと起きる「わるいこと」

5.3.1 サイトが「安全でない」と表示されてしまう

GoogleはHTTPからHTTPSへの移行を強く推進しています。その施策の1つとして、Googleが提供するブラウザの「Chrome」では、2018年7月にリリースされた「Chromeバージョン68」から、HTTPSでないページに対して「保護されていない通信」という表示をするようになりました。

Chromeでサイトを開いて、URLが「http://」で始まるものだった場合、次のようにURLの左側に「保護されていない通信」と表示（図5.1）されます。

① 保護されていない通信 | ikinaristeak.com/home/

▲図 5.1 いきなり！ ステーキのサイトを開くと「保護されていない通信」と表示される

サイトを開いたときに、URL の真横に「保護されていない通信」と表示されたら、「何か危なそう…見ない方がいいかも…？」と心配になってしまいですね。Chrome だけではなく、同様に Firefox も錠前に赤い斜め線が入ったマークの表示（図 5.2）を行っており、サイトを HTTPS 化しないことで、エンドユーザにサイトが「安全でない」と思われてしまう状況にあります。



▲図 5.2 HTTP のサイトを開くと「保護されていない通信」と表示される

5.3.2 Wi-Fi スポットでセッションハイジャックされる恐れがある

Amazonなどのサイトを開くと、まだログインしていないのに、ログイン済みのページが表示されることがありますよね？ これは過去にログインした際に、サイトから Cookie で渡された「セッション ID」（一時的な通行証のようなもの）を、次に開いたときにも提示することで、ログイン済みのユーザーとして扱われているからです。

ログインページだけが HTTPS のサイトだと、それ以外のページを HTTP で開いたとき、Cookie に Secure 属性が付いていないと、この「セッション ID」は暗号化されない状態で送信されてしまいます。では、誰でも使える Wi-Fi スポットに、スマホやタブレットを繋いだ状態で、HTTP のページを開いて「セッション ID」が送られたらどうなるでしょう？

なんと同じ Wi-Fi につないでいる悪意の第三者によって、暗号化されていない「セッション ID」を盗まれ、なりすましてサイトにログインされる恐れがあります。これが「セッションハイジャック」と呼ばれる攻撃です。

こうした攻撃で不正にログインされないよう、サイト全体を HTTPS にして、常に Cookie の「セッション ID」を暗号化しておかないと危険です。またログインなどが一切ない、完全に静的コンテンツのみのサイトであっても、Wi-Fi スポットで見知らぬ誰かに「あの人はどんなサイトを見ているのかな？」とデータを窺視されるリスクもあります。

例えば「妊活情報のブログを長時間読んでいた」「特定の病気について調べていた」など、どんなサイトを見ていたのか？という情報の中にも、人に知られたくないことはたくさんあります。サイトとの通信が HTTPS で暗号化されていれば、情報を盗み見られるエンドユーザにとってのリスクを防ぐことができます。

5.3.3 相対的に検索順位が下がる

さらに、インターネット全体で HTTPS 化が進むことによって、HTTP のままでいるサイトに起きるデメリットもあります。

Google は「HTTPS everywhere」、つまり「(お問い合わせや会員登録といった一部のページに限らず) どこでも HTTPS！」を提唱しています。2014 年 8 月の時点で既に、HTTPS に対応しているウェブサイトを検索ランキングで優遇する方針も発表^{*1}しています。

Google のランキングアルゴリズムで、「サイトが HTTPS 化されているか」が指標のひとつとなっています。もちろんたくさんある指標の中のひとつで、他の指標ほどウェイトは大きくない、とされていますが、競合サイトの HTTPS 化が進めば相対的に検索順位が下がる可能性があります。

5.3.4 周りが HTTPS になるとリファラーが取れなくなる

こちらは Google アナリティクスなどを使って、サイト流入元の情報を確認している方にとって、重要なデメリットです。

たとえば自社のサイトが HTTP だと、HTTPS のサイトからリンクを踏んで飛んできた場合に、リファラ（利用者が直前に訪問していたサイトの情報）を取得することができません。実際に Google アナリティクスを開いて、集客の「参照元/メディア」を確認してみてください。アクセス元が「(direct) / (none)」と表示されて、どこから飛んできたのか分からぬものはありませんか？その中には、ブラウザのブックマークや、メール内のリンクから飛んできた、本当に「直前に訪問していたサイトが存在しないもの」だけでなく、HTTPS のサイトから飛んできたアクセスも含まれています。

今後、周囲のサイトの HTTPS 化が進んで、自社のサイトだけが HTTP で取り残されると、この「リファラーが取得できる割合」はますます下がっていくことになります。自社のサイトを HTTPS にすれば、HTTP のサイトから飛んできた場合も、HTTPS のサイトから飛んできた場合も、リファラーを取得することができるようになります。

^{*1} Google ウェブマスター向け公式ブログ \[JA]: HTTPS をランキング シグナルに使用します https://webmaster-jp.googleblog.com/2014/08/https-as-ranking-signal.html

5.4 HTTPS 化すると起きる「いいこと」

5.4.1 表示速度が上がる

かつては、HTTPS にすると、暗号化と復号のオーバーヘッドのためウェブサイトが遅くなるというのが常識でした。しかし、サーバ側、クライアント側ともに CPU の性能が向上したこと、2020 年現在、もはやオーバーヘッドは重要視するほどではなくなっています。^{*2}

それだけでなく、HTTPS にすることで HTTP/2 というプロトコルを利用できるようになり、むしろ表示の速度が速くなるケースもあります。

5.4.2 Same Site 問題に対応できる

Chrome は 2020 年 2 月にリリースされる Chrome 80 以降、クロスサイトの Cookie は「SameSite=None; Secure」の設定がなければ、アクセスしたサイトの Cookie として保存・利用できなくなります。Secure 属性を追加するには、HTTPS での接続が必須です。

<https://developers-jp.googleblog.com/2019/11/cookie-samesitenone-secure.html>

5.4.3 大事な情報のアタリが付けにくくなる

流れていくデータの殆どが平文な中で、たまに暗号化されたデータが流れてくると、データを眺めていた悪意の第三者は「暗号化されてるってことは、あれは大事な情報だな！」とアタリを付けることができます。

ですが、すべてのサイトが HTTPS になって、流れてくるデータがすべて暗号化されていれば、どれが重要な重要なデータなのかアタリが付けられなくなります。大事なものも大事でないものもすべて同じ頑丈なアタッシュケースにしまって運べば、泥棒はどのアタッシュケースを狙えばいいのか分からなくなります。木は森に隠せ、という戦法ですね。

^{*2} サーバの手前に設置して、暗号化や復号だけを行なう専用機器の「SSL アクセラレータ」を使う環境の話も、最近はほぼ聞かなくなりました

5.5 SSL/TLSとは？

HTTPS化するメリット、しないデメリットが分かったところで、改めて「SSL/TLSってなに？」というところを学んで行きましょう！

SSL（Secure Socket Layer）/TLS（Transport Layer Security）とは、インターネット上で安全にデータを送受信するための約束事（プロトコル^{*3}）です。

SSLもTLSもあくまでプロトコル、つまり通信するための「約束事」なので、実際に通信するときは、そのプロトコルに従って実装されたソフトウェアを使います。SSL/TLSでは、OpenSSLというオープンソースのソフトウェアを使うことが殆ど^{*4}です。

5.5.1 SSLとTLSはどういう関係？

SSLとTLSは別々の名前ですが、その役割に大きな違いはありません。「SSL3.0」からバージョンアップする際に、「SSL3.1」ではなく「TLS1.0」という新しい名前が付けられました。つまり「TLSはSSLの後継バージョン」ということです。

ちなみに名前がまだSSLだったときの最後のバージョン「SSL3.0」は、重大な脆弱性^{*5}が見つかり、2015年のRFC7568^{*6}でもう使わないよう「Do Not Use SSL Version 3.0」と示されています。

ですがSSLという言葉の認知度が高く、TLSと呼んでもピンとこない人の方が多いため、現状は分かりやすさと正確さを両立するためにSSL/TLSと併記されることが多いです。

本著ではSSL/TLSのことを指して、SSLという言葉を使用します。

5.5.2 SSLイコールHTTPSではない

サイト全体を「https://」から始まるURLにすることを、「常時SSL化」のように呼ぶため、皆さんの中にはSSL=HTTPSだと思っている人もいるかも知れません。

HTTPとSSLを組み合わせて使うことで、通信を保護するプロトコルがHTTPS

^{*3} 例えば手紙は「メッセージを書いた便せんを封筒に入れて、表に宛先、裏に差出人を書き、重さに応じた切手を貼ってポストに入れる」という取り決めに従えば、きちんと相手に届きます。手紙の例と同じように、インターネットで通信を行う際、どんなデータをどんな方法で送受信するのか、という手順の約束事のことをプロトコルと呼びます。SSLもTLSもプロトコルですし、HTTPやHTTPS、DNSもSMTPもプロトコルです

^{*4} さっき証明書を取得するときに打ったコマンドも、opensslコマンドでしたね

^{*5} 脆弱性（ぜいじやくせい）というのは悪用が可能なバグや設定不備のことです

^{*6} Deprecating Secure Sockets Layer Version 3.0 <https://tools.ietf.org/html/rfc7568>

(HTTP over SSL/TLS) です。ですが、SSL は HTTPSにおいてのみ使われるプロトコルではありません。例えばサーバにファイルをアップするときの FTP と組み合わせて使う FTPS (FTP over SSL) や、メール送信の SMTP と組み合わせて使う SMTPS (SMTP over SSL) など、HTTPS 以外にも SSL が使われている場面はいろいろあります。

繰り返しになりますが、SSL はインターネット上で安全にデータを送受信するためのプロトコルです。上位のアプリケーション層^{*7}が HTTP であれ、FTP であれ、SSL と組み合わせて使うことでデータを安全に送受信できるようになります。

5.6 SSL 証明書とは

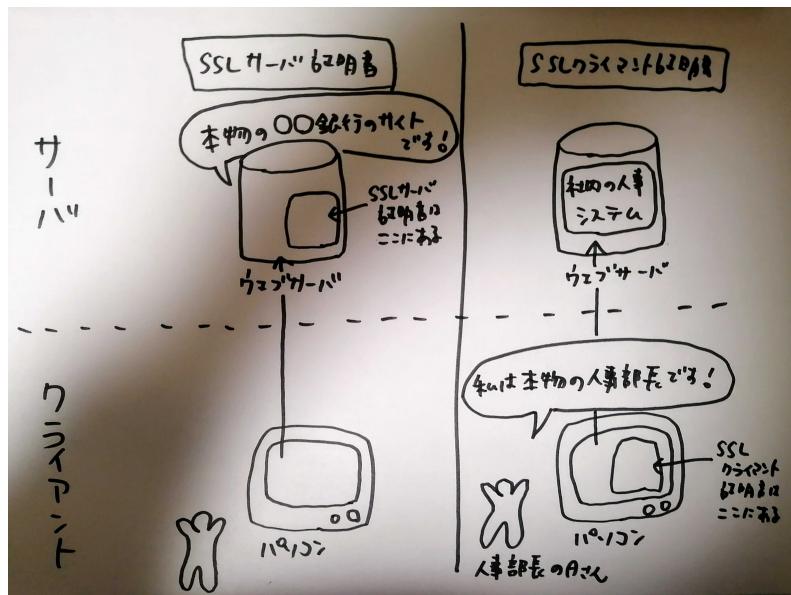
続いて、HTTPS 化するため必要な SSL 証明書について学んでいきましょう。

5.6.1 SSL サーバ証明書と SSL クライアント証明書

「SSL 証明書」という名前を聞いたことはありますか？名前だけは知っているという人も、実際に仕事で使ったことがあるという人もいると思います。

SSL 証明書には、SSL サーバ証明書と SSL クライアント証明書の 2 種類（図 5.3）があります。ざっくり言うとサーバの身元を証明するのが SSL サーバ証明書で、クライアントの身元を証明するのが SSL クライアント証明書です。

^{*7} OSI 参照モデルのアプリケーション層のこと。エンジニア諸氏は誰しも、大学の授業や新卒研修で一度は「アセトネデブ」という語呂合わせを聞いたことがあるのでは…



▲図 5.3 SSL サーバ証明書と SSL クライアント証明書

単に「SSL 証明書」という略称で呼んだときは、「SSL サーバ証明書」のことを指すことが殆どです。本著でも SSL サーバ証明書のことを指して、SSL 証明書という言葉を使用します。

5.6.2 SSL 証明書はどんな場面で使われている？

SSL 証明書はどんな場面で使われているのでしょうか？

SSL 証明書は、あなたがブラウザで「https://」から始まる URL のサイトを開いて、次のようなマーク（図 5.4、図 5.5）が表示されているときに使われています。

 techbookfest.org/event/tbf08

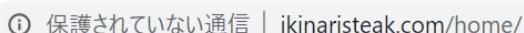
▲図 5.4 Chrome で HTTPS のサイトを開いたとき

5.7 SSL 証明書は全然違う 2 種類の仕事をしている



▲図 5.5 Firefox で HTTPS のサイトを開いたとき

逆に「http://」から始まる URL のサイトを開いて、次のようなマーク（図 5.6、図 5.7）が表示されているときは使われていません。



▲図 5.6 Chrome で HTTP のサイトを開いたとき



▲図 5.7 Firefox で HTTP のサイトを開いたとき

5.7 SSL 証明書は全然違う 2 種類の仕事をしている

「SSL 証明書は https://から始まるページで使われている」ということが分かったところで、次に、https://から始まるページにおいて、SSL 証明書は一体何をしているのでしょうか。実は、SSL 証明書は全然異なる次の 2 つの仕事をしています。この点が SSL 証明書の話の分かりにくさの原因なのです。

- ・ ウェブサイトで送受信する情報を暗号化すること
- ・ ウェブサイト運営者の身元を証明すること

それでは 1 つ目の役割、「 ウェブサイトで送受信する情報を暗号化すること」をから、詳しく説明していきます。

5.7.1 ウェブサイトで送受信する情報を暗号化すること

皆さんが HTTP から始まる URL をブラウザで開いたとき、入力した URL の文字やサーバから届く Web ページ、画像などは何も暗号化されていません。パソコンからサー

バまでのリクエスト（往路）も、サーバからパソコンまでのレスポンス（復路）も、そのままの姿でだーっと流れていきます。

データが暗号化されないため、パソコンとサーバの間のネットワークを盗聴すれば、あなたが入力して送った文字や表示された画像は、誰でも盗み見ることができます。

もしこれが Amazon や Twitter にログインするときだったらどうでしょう？ 入力した ID やパスワードがそのままの姿で流れていったり、過去の購入履歴が Amazon のサーバから自分のパソコンまで流れたりして、途中誰でも盗み見られる状態だったら困りますよね。

そこで出てくるのが HTTPS です。HTTPS ならネットワーク上を流れるデータが暗号化されます。分かりやすく言うと、ブラウザで HTTPS から始まる URL を開いたとき、自分のパソコンとサーバの間を流れるデータのすべては SSL 証明書によって暗号化されます。

これは送信も受信も両方ですので、ログインやファイルアップロードなどの送信のときも、購入履歴画面や会員情報確認画面など、自分の情報をサーバから受け取って表示する受信のときも、データはすべて暗号化されます。データが、中の見えない（透明ではない）安全なトンネルを通っていくようなものだと思ってください。

SSL 証明書がないと、HTTPS から始まる URL でページを開くことはできません。これが SSL 証明書の 1 つ目の役割、「 ウェブサイトで送受信する情報を暗号化すること」です。このような理由から昨今、エンドユーザに情報を送ったり、受け取ったりするサイトでは、SSL 証明書は必須となっています。

5.7.2 ウェブページは 1 往復で表示されるわけじゃない

1 つのウェブページを表示するときの流れは、「トップページをください」「はい完成品をどうぞ」の一往復だけではありません。次のように、何往復ものリクエストとレスポンスの末にページが表示されます。

- ・「example.com さん、ページをください」「はい、HTML どうぞ」
- ・「cdn.example.com さん、base.css をください」「はい、base.css どうぞ」
- ・「cdn.example.com さん、common.js をください」「はい、common.js どうぞ」
- ・「cdn.example.com さん、top.png をください」「はい、top.png どうぞ」
- ・「cdn.example.com さん、banner01.jpg をください」「はい、banner01.jpg どうぞ」

この 1 行 1 行がリクエストとレスポンスの往復を表しているのです。

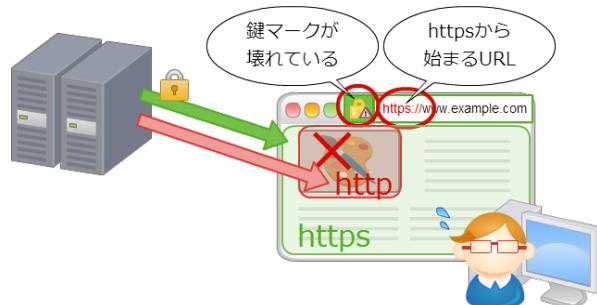
5.7.3 画像と CSS の指定が絶対パスだった

「HTTPS から始まる URL を開いたときに鍵マークではなく i マークが表示される」現象の原因も、この暗号化が関係しています。

ブラウザで HTTPS から始まる URL を開くと、Web ページはサーバから自分のパソコンまで、HTTPS で暗号化されて届きます。しかし取得したページの HTML に、``のような、絶対パスで画像を指定する``タグが含まれていると、その画像ファイルは暗号化されていない HTTP で送られてきます。

つまり、せっかく Web ページを HTTPS で開いても、その Web ページの HTML ソースの中で、CSS や画像ファイルの指定を `http://`から始まる形式にしてしたり、YouTubeなどのコンテンツを `http://`から始まる形式で埋め込んでいると、ブラウザは鍵マークの代わりに i マークを表示して「ページの一部が暗号化されていないので、安全ではありません」と警告を出すのです。

i マークをクリックすると、「このサイトへの接続は完全には保護されていません」と表示されます。



▲図 5.8 混在コンテンツ（mixed content）はエラーが出る

これを直すには、``タグを ``のようにパスの部分だけにします。これならページを HTTP で開いたときは画像も HTTP で表示でき、HTTPS で開いたときは画像も HTTPS で表示できるため、鍵マークが壊れることはありません。

あるいは、「☒画像は Web ページとは別のサーバにあるので、ドメインから書かなければいけない」という場合は、``タグを ``のように書くことで、先ほどと同じように、ページを HTTP で開いたときは画像も

HTTPで表示でき、HTTPSで開いたときは画像もHTTPSで表示できます。その結果、鍵マークは壊れません。

テンプレートファイルなどをHTTP/HTTPS両方のページで利用しているケースでは、後者の「//から始まる書き方」にしておくと、鍵マークが壊れる現象は避けることができてお勧めです。

5.7.4 ウェブサイト運営者の身元を証明すること

役割の話に戻って、2つ目の役割「ウェブサイト運営者の身元を証明すること」について詳しく説明します。

この役割を一言で説明するなら、SSL証明書は「ウェブサイト運営者の身元証明書」であるということです。このことを分かりやすく説明するために、架空の銀行を舞台にした、たとえ話をします。

5.7.5 ネットバンクの事例

あなたは、立ち上げたばかりの株式会社 EXAMPLE 銀行の広報担当者です。EXAMPLE 銀行の「bank.example.co.jp」というドメインを取得し、無事ウェブサイトを開設しました。EXAMPLE 銀行は店舗を持たないネットバンクで、ブラウザで bank.example.co.jp を開き、ID とパスワードを入れてログインすると、残高の確認や振込、キャッシングなどをすることができます。

そこへ目をつけた詐欺師がいました。詐欺師は bank.example.jp というよく似たドメインを取得して、bank.example.co.jp にそっくりな偽サイトを作りました。ウェブサイトは基本的に「公開」されているものなので、丸ごとコピーしてパクリサイトを立ち上げることは意外と簡単です。そのため、サイトの見た目だけなら、bank.example.co.jp（本物サイト）と bank.example.jp（偽サイト）はそっくりで区別がつきません。

そして詐欺師は EXAMPLE 銀行の利用者向けに、EXAMPLE 銀行をよそおって次のメールを出しました。

「EXAMPLE 銀行より緊急のご連絡です。お客様の口座で不審な取引が確認されました。いますぐに下記のリンクからログインして、お客様の口座残高をご確認ください。24時間以内に確認を行わなかった場合、お客様の口座は凍結されます」

メールを見た利用者が慌ててリンクを踏んで、bank.example.jp（偽サイト）を開いてしまい、そこでIDとパスワードを入力すると、詐欺師はIDとパスワードを盗むことができます。そしてその盗んだIDとパスワードで、詐欺師が bank.example.co.jp（本物サイト）にログインしたら、あとは簡単ですよね。限度額までキャッシングをして、すべて

詐欺師の口座に振り込めば、大金が一瞬で詐欺師のものになるのです。

このように偽サイトへ誘導して ID やパスワードを盗む詐欺の手法を「フィッシング詐欺」と言います。自社のお客様がこんな目にあったら大変ですよね。誰も EXAMPLE 銀行を使ってくれなくなってしまいます。急いで対策を立てて、フィッシング詐欺からお客様を守らなければなりません！

お客様からの問い合わせで事態を知った EXAMPLE 銀行広報担当のあなたは、慌てて bank.example.co.jp (本物) のトップページに「bank.example.jp は EXAMPLE 銀行を騙った悪質な偽サイトです。ご注意ください」とお知らせを出しました。しかし、なんと bank.example.jp (偽物) のトップページにも「bank.example.co.jp は EXAMPLE 銀行を騙った悪質な偽サイトです。ご注意ください」と書かれてしまいました。

トップページに注意書きを出したが効果なし

画像

もう EXAMPLE 銀行の利用者は、どちらを信じていいのか、どちらが本物の EXAMPLE 銀行のサイトだか分かりません。店舗があれば窓口で「EXAMPLE 銀行のサイトは bank.example.co.jp です！」と書いたチラシを配ることもできますが、EXAMPLE 銀行はネットバンクなので店舗がありません。ネットだけで「bank.example.co.jp こそが EXAMPLE 銀行が運営する本物のサイトだ」と証明するには、一体どうしたらいいのでしょうか？

ここで登場するのが SSL 証明書です。

先ほど述べたとおり、SSL 証明書は「ウェブサイト運営者の身元証明書」です。サイバートラストやシマンテック（シマンテックは少し前までペリサインというブランド名でした）といった「認証局事業者」に対して、EXAMPLE 銀行広報担当のあなたが「bank.example.co.jp の SSL 証明書を発行してください」と依頼すると、

株式会社 EXAMPLE 銀行という会社が実在するのか？ bank.example.co.jp というドメインの所有者は証明書の発行を承認しているのか？ 証明書の発行依頼をしてきたのは株式会社 EXAMPLE 銀行なのか？ の 3 点をよくよく調査・確認した上で、「bank.example.co.jp は株式会社 EXAMPLE 銀行が運営するサイトです」という身元証明書を発行してくれます。

この身元証明書こそが、みなさんご存知の「SSL 証明書」なのです。

この SSL 証明書をウェブサイトに設置しておくと、エンドユーザがブラウザでサイトにアクセスしたとき、サイトのデータと一緒に、身元を証明する SSL 証明書が送られてきます。本物のサイトには「認証局事業者が発行してくれた身元証明」があるため、HTTPS から始まる URL を開くと URL の左側に鍵のマークが表示されます。一方、詐欺師が認証局事業者に「オレ、EXAMPLE 銀行。SSL 証明書発行してよ」と言っても、EXAMPLE 銀行ではないので、先ほどの 3 点が確認できません。よって SSL 証明書は発

行してもらえず、鍵マークが出ないどころか、HTTPS から始まる URL でサイトを開くことすらできません。

SSL 証明書があればウェブサイト運営者の身元証明ができる
画像

これで無事に「bank.example.co.jp こそが EXAMPLE 銀行が運営する本物のサイトだ！」と証明することができました。ということで、繰り返しになりますが、SSL 証明書は「ウェブサイト運営者の身元証明書」になるのです。

ちなみにこの「証明」とは、

株式会社 EXAMPLE 銀行という会社が実在すること bank.example.co.jp というサイトの運営者が株式会社 EXAMPLE 銀行であることを証明しているだけで、

株式会社 EXAMPLE 銀行が社会的に信頼に足る会社であること株式会社 EXAMPLE 銀行が経営状態のよい優良企業であることなどは一切証明していません。身元証明といつても、いわゆる保証人のような身元保証ではないので、その点は間違えないよう注意してください。

さて、話を進めます。フィッシング詐欺に失敗した詐欺師は、「☒失敗したか。でもウェブサイトを丸ごとパクったみたいに、SSL 証明書も盗んでくればいいや！」と思いました。

しかしウェブサイトと違って、SSL 証明書はそう簡単には盗めないようになっています。しかもなんとか頑張って盗んできたとしても、その SSL 証明書は「bank.example.co.jp に対する証明書」なので、「☒ bank.example.jp」のサイトに設置すると、ブラウザで次のようなエラーメッセージが表示されます。

bank.example.co.jp の SSL 証明書を bank.example.jp で使うと出るエラーメッセージ
画像

つまり、SSL 証明書は盗んでも意味がないのです。このように SSL 証明書という身元証明を使って、HTTPS でページを表示すれば、詐欺師が身元を騙ることは途端に難しくなります。

これが SSL 証明書の 2 つ目の役割、「☒ ウェブサイト運営者の身元を証明すること」になります。

5.8 認証局事業者の身元は誰が証明する？

このたとえ話では、認証局事業者が「bank.example.co.jp は株式会社 EXAMPLE 銀行が運営するサイトです」という SSL 証明書を発行してくれたので、EXAMPLE 銀行のウェブサイトは運営者の身元を証明できましたとさ。めでたしめでたし、ということになります。

でもちょっと待ってください。サイバートラストやシマンテックといった「認証局事業者」は本物なのでしょうか？

詐欺師 A が EXAMPLE 銀行になりすまし、協力する詐欺師 B が認証局事業者になりますとして、適当な SSL 証明書を自作しただけかもしれません。疑えばきりがありませんが、全然知らない A さんに、全然知らない B さんの身元保証をしてもらっても、いまいち信用できないのと同じ話です。

そこで登場するのが「中間 CA 証明書」です（CA は Certificate Authority の頭文字で、日本語に訳すと認証局事業者です☒）☒。

認証局事業者にお金を払って SSL 証明書を発行してもらうと、「☒中間 CA 証明書」という証明書が一緒にきます。SSL 証明書はウェブサイト運営者の身元証明書でしたが、中間 CA 証明書は認証局事業者の身元証明書なのです。

実は SSL 証明書による「身元証明」は、「☒最下層の SSL 証明書>中間 CA 証明書>最上位のルート証明書」とピラミッドのように階層化されています。

SSL 証明書？ 中間 CA 証明書？ ルート証明書？ ピラミッドのように階層化？ いろいろな名前が出てこんがらがってきました。実例を元に説明したほうが分かりやすいので、株式会社イグザンブルのコーポレートサイトを例にして、この身元証明の連鎖を見てみましょう。

5.8.1 身元保証の連鎖をつなぐ中間 CA 証明書とルート証明書

株式会社イグザンブルのコーポレートサイトには、サイバートラストから発行された SSL 証明書と中間 CA 証明書が設置されています。

ここで、サイバートラストが提供している「SSL 証明書の設定確認ツール」を使って、SSL 証明書を見てみましょう。

サイバートラストの「SSL 証明書の設定確認ツール」

画像

「SSL 証明書の設定確認ツール」を開いたら、フォームに「techbookfest.org」と入力し、青いボタン（設定を確認する）を押してみましょう。すると、次のような画面が表示されます。

techbookfest.org の SSL 証明書設定が表示された

画像

SSL サーバ証明書と中間 CA 証明書 1、2 のそれぞれの赤いボタンを押せば、詳細情報を表示できます。いろいろなことが記載されていますが、要約すれば次の表に挙げたことが書かれています。

SSL サーバ証明書 techbookfest.org の運営者「EXAMPLE Corporation」の身元は

「Cybertrust Japan Public CA G3」が保証する中間CA証明書1 SSLサーバ証明書を発行した「Cybertrust Japan Public CA G3」の身元は「Baltimore CyberTrust Root」が保証する中間CA証明書2 中間CA証明書1を発行した「Baltimore CyberTrust Root」の身元は「GTE CyberTrust Global Root」が保証する身元保証の連鎖ですね。株式会社イグザンブルを仮にAさんとすると、Aさんの身元をBさんが、Bさんの身元をCさんが、Cさんの身元をDさんが保証しています。

SSL証明書と中間CA証明書はサーバに設置されているため、ブラウザでサイトを開くと、Webページと一緒にパソコンへ届けられます。これでAさん、Bさん、Cさんの身元は保証されましたが、ではDさんこと一番下の中間CA証明書2を発行した「GTE CyberTrust Global Root」の身元を証明する「ルート証明書」はどこにあるのでしょうか。

実は、ルート証明書は皆さんのが使っているブラウザに最初から入っているのです。ChromeやFirefox、IEといった各ブラウザには、身元保証の連鎖の頂点にある「ルート証明書」が最初からインストールされています。

Chromeなら、設定(S) > 詳細設定を表示... > HTTPS/SSLの「証明書の管理...」をクリック>「信頼されたルート認証機関」のタブをクリックすると、そこにDさんこと「GTE CyberTrust Global Root」の身元を証明する「ルート証明書」があることを確認できます。

Chromeに入っているルート証明書のリスト

画像

ちなみにルート証明書の「発行先」と「発行者」を見ると、まったく同じ「GTE CyberTrust Global Root」です。これはDさんの身元はDさん自身が保証し、それをブラウザが信頼したので、ここに最初からルート証明書が入っているという図式です。

このようにSSL証明書と中間CA証明書がWebサーバに置いてあり、そして「ルート証明書」がブラウザに入っているため、株式会社イグザンブルのコーポレートサイトは「株式会社イグザンブルが運営する本物のサイト」として信頼してもらえるのです。

身元保証連鎖の頂点にあたるルート証明書はブラウザに入っている

画像

SSL証明書ってこんな仕組みになってたんですね。

5.9 SSL証明書はどうしてみんなに値段に差があるの？

ではSSL証明書の仕組みが分かったところで、SSL証明書における最もありがちな疑問、「どうしてSSL証明書はみんなに値段に差があるの？」についても、少しお話しします。

SSL 証明書で検索してみると、シマンテックは 219,000 円、サイバートラストは 75,000 円、そしてサイフにやさしい SSL 証明書こと RapidSSL は 2,600 円でした（執筆当時）
※。この価格差は驚きますよね。

なぜこんなに価格差があるのでしょうか。シマンテックはブランド代が含まれるので高いのでしょうか？ あるいは RapidSSL が企業努力の塊なのでしょうか？ それとも同じ「SSL 証明書」という名前でも、RapidSSL の SSL 証明書は中身が何か違うのでしょうか？

5.10 同じ「SSL 証明書」という名前でも 3 つの種類がある

結論から言うと違います。

SSL 証明書の役割は次の 2 つであることを説明してきました。

ウェブサイトで送受信する情報を暗号化することウェブサイト運営者の身元を証明することしかし、RapidSSL が発行しているような安い SSL 証明書は、ウェブサイト運営者の身元証明をせず、「情報の暗号化」だけしか行いません。

「え、それって SSL 証明書だって言えるの？」と思うかもしれません、言えます。なぜなら、SSL サーバ証明書には実は 3 つの種類があるからです。

「SSL 証明書」とひとくちに言っても、その実態は 3 種類に分かれています。分かりやすくいうと「高い」「普通」「安い」の 3 種類で、それぞれ「EV 証明書」「 OV 証明書」「 DV 証明書」という名前です。

5.10.1 3 つの違いは何か？

種類 何を証明してくれる？ 商品例 EV 証明書 (Extended Validation) ウェブサイト運営者の身元をより厳格に書類と電話で確認して証明 ・ サイバートラストの SureServer EV ・ シマンテックのセキュア・サーバ ID EV OV 証明書 (Organization Validation) ウェブサイト運営者の身元をメールと電話で確認して証明 ・ サイバートラストの SureServer ・ シマンテックのセキュア・サーバ ID DV 証明書 (Domain Validation) そのドメインの使用権があることを証明 ・ RapidSSL RapidSSL のような DV 証明書は、SSL 証明書という名前で呼ばれていますが「ウェブサイト運営者の身元証明」は一切行いません。

5.10.2 DV 証明書

ドメインの所有者を「Whois」と呼ばれるドメインの登録者情報で確認し、そこに書いてある所有者のメールアドレスに対して「このドメインの SSL 証明書を発行していいですか？」と確認てくるだけです。ですから、ドメイン所有者が発行承認ボタンを押した

ら、それだけですぐに発行されます。

このように DV 証明書は、「 ドメインの使用権があること」の確認と証明をするだけで、「 誰がそのウェブサイトを運営しているのか?」という身元確認及び身元証明はしてくれません。したがって DV 証明書は、先ほどの EXAMPLE 銀行のように「身元の証明をしたい」ケースでは使う意味がありません。HTTPS で暗号化はしたいけれど、身元証明をする必要度はあまりない、開発用のテスト環境等で使用されることが多いです。

5.10.3 EV 証明書と OV 証明書

DV 証明書の役割は分かりました。では残りの 2 つ、「 高い = EV 証明書」と「普通 = OV 証明書」の違いは何なのでしょうか。先ほど例として挙げた株式会社イグザンブルのコーポレートサイトでは、「 普通」の OV 証明書を採用しています。

EV 証明書と OV 証明書は、DV 証明書と違って、SSL 証明書の 2 つの役割をきちんと果たします。

しかし OV 証明書は、ブラウザの鍵マークをクリックして証明書を開き、その中の証明書情報を確認しないと、サイト運営者の名前が表示されません。分かりにくいくらいで、実際の画面を見てみましょう。例えば Firefox で見たとき、株式会社イグザンブルのコーポレートサイトのお問い合わせ画面は次のようにになります。

驚かれるかもしれません、OV 証明書の場合、ここにウェブサイトの運営者名は表示されず、「 このサイトの運営者は不明です」になってしまいます。なぜならば「ウェブサイト運営者の身元をメールと電話で確認しただけで、確認度合いが低いため、身元証明がいまいち信用できない」とブラウザ (Firefox) が思っているからです。

鍵のマークをクリックした後に、「 詳細を表示...」をクリックし、さらに「証明書を表示...」をクリックすると、ここで初めて運営者名として「EXAMPLE Corporation」という名前が出てきます。EV 証明書のように、「 ウェブサイト運営者の身元を書類と電話でより厳格にチェックして証明」されたものでないと、ここには運営者の名前は出ないです。

Firefox の証明書詳細情報確認画面

画像

これは結構重要な問題です。つまり一見しただけなら、身元を証明しない DV 証明書と、身元を証明する OV 証明書は区別がつかないので、先ほどの EXAMPLE 銀行の例に戻ってみましょう。EXAMPLE 銀行の広報担当が身元証明をする OV 証明書を取得して、詐欺師が身元を証明しない DV 証明書を取得した場合、次のようにになります。

OV 証明書と DV 証明書はぱっと見ただけでは区別がつかない

画像

5.10 同じ「SSL 証明書」という名前でも 3 つの種類がある

もちろん、証明書の情報をよく確認してもらえば、片方が身元証明されていないことは分かるのですが、そんなことをするエンドユーザは滅多にいません。

そこで、ぱっと見ただけで「偽物と区別がつくようにしたい」「成りすましを防ぎたい」というときは、DV 証明書でも OV 証明書でもなく、EV 証明書を使う必要があります。実際、三井住友銀行や三菱東京 UFJ 銀行を始めとする国内のネット銀行は、ほとんどが EV 証明書を採用しています。

シマンテックの EV 証明書を採用している三菱東京 UFJ 銀行のサイト

画像

またネットショップのように、クレジットカード情報を入力するサイトでも、EV 証明書を採用するところが増えました。例えば「山田養蜂場」というはちみつや自然食品のオンライン販売をしているサイトでは、サイバートラストの EV 証明書を使っているため、URL の左側にサイトの運営者名が日本語で出ています。また鍵マークをクリックすると、社名に加えて住所も表示されるため、どこのだれが運営しているサイトなのかがすぐに分かります。

EV 証明書を採用している山田養蜂場のサイト

画像

ネット銀行やネットショップなど、偽物が出やすく、かつ偽物による被害が出た場合にダメージが大きいサイトでは、多少値段が高くても EV 証明書を使う意味があるということなのです。

このように「SSL 証明書」という 1 つの名前でも、その中で「DV 証明書」「 OV 証明書」「 EV 証明書」の 3 種類に分かれています。値段が大きく違います。高い SSL 証明書と安い SSL 証明書がある理由が納得できましたか？

5.10.4 さよならグリーンバー

2019 年 9 月にリリースされた Chrome バージョン 77、そして翌月 10 月にリリースされた Firefox バージョン 70 からは、グリーンバーは表示されなくなりました。

5.10.5 ブラウザベンダーによるEV証明書の扱いの変化

5.11 その他の証明書

5.11.1 中間証明書

5.11.2 クロスルート証明書

5.12 どの証明書を買えばいい？

5.12.1 ワイルドカード証明書

5.12.2 wwwありにリダイレクトしたいだけなのにwwwなしの証明書もいるの？

5.12.3 コモンネームが*.example.comの証明書はexample.comで使える？

5.12.4 SANs

5.12.5 Let'sEncrypt

5.13 CDNと証明書

5.13.1 CDNを使ったら古い端末でサイトが見られなくなった

5.13.2 同じサーバで複数サイトをHTTPS化したら古い端末で別サイトが表示された

5.13.3 SNI Server Name Indication

HTTPSで使われるTLSプロトコルでは、接続したいホスト名をクライアント側からサーバに伝えるためにSNI(Server Name Indication)のTLS拡張が必要となります。ただしSNIは、1つのIPアドレスを複数のバーチャルホストで共用するため、HTTPSで使用した場合、SNI非対応のクライアントではデフォルトのホストが応答します。2019年現在、SNI非対応端末を「対象端末」としているサービスはあまり多くないかもしれません。

あとがき

好きな技術の本を書くのは、とても楽しいものです。「誰に頼まれた訳でもないのになぜ書くの？」と自分に問いかけると、そこには、「これを書いたら、きっと誰かの助けになるはずだ」という祈りのような希望があります。

祈りのはかなさに対して、自覚すらない悪意はとても強いです。第三者から気軽にぶつけられた石の痛みに呆然としながら、「もう書くのやめようかな」と思うときもあります。

以下は、そんな筆者に対して、同僚であり友人でもある Marshall が送ってくれたメッセージです。何かを書いたり、作ったりして、誰かの役に立とうとしている人、みんなの胸に灯りをともすようなメッセージだったので、あとがきに載せる許可をもらいました。

快く承諾してくれた Marshall に感謝します。

To be honest, people can find **ANY** information on the internet. Everything I learned at my university I could have learned from the internet--I still went to university.

There is a lot of information floating around online. There is a lot of information about marketing, but I still have a ton of books at home on how to become a better marketer.

People can take pictures with their smartphone--that doesn't mean people who draw should stop drawing or people who paint should stop painting.

Keep writing!

I've written a lot of articles online with the intention of helping people. People will leave comments like "What a stupid article! You think people don't already know how to do this? Why waste your time writing this!?"

But my answer is because I know my article helped someone and that makes me happy. Not everyone knows everything and if I can share my knowledge and help at least one person, then it is worth it for me.

The point I'm trying to make is no matter what you do there is someone who will try to make you feel bad about it. Keep doing it anyway.

数ある技術書の中から「SSLをはじめよう」を手に取ってくださったあなたに感謝します。

2020年3月
mochikoAsTech

PDF版のダウンロード

本著（紙の書籍）をお買い上げいただいた方は、下記のURLからPDF版を無料でダウンロードできます。

- ダウンロード URL : <https://mochikoastech.booth.pm/items/xxxxxx>
- パスワード : **xxxxxx**

Special Thanks:

- Gunnell Marshall

レビュアー

- Takeshi Matsuba
- Mari Kubota

参考文献・ウェブサイト

- プロフェッショナル SSL/TLS Ivan Ristić、齋藤孝道（監訳）
 - <https://www.lambdanote.com/products/tls>
- プロフェッショナル IPv6 小川晃通
 - <https://www.lambdanote.com/products/ipv6>
- 食べる！SSL！—HTTPS環境構築から始めるSSL入門 小島拓也、中嶋亜美、吉原恵美、中塚淳
 - <https://www.amazon.co.jp/dp/B00PHC4480>
- SSL/TLSの基本 - Qiita
 - https://qiita.com/angel_p_57/items/446130934b425d90f89d

-
- 【図解】初心者も分かる”公開鍵/秘密鍵”の仕組み～公開鍵暗号方式の身近で具体的な利用例やメリット～ | SE の道標
 - <https://milestone-of-se.nesuke.com/sv-advanced/digicert/public-private-key/>
 - 「電子署名=『秘密鍵で暗号化』」という良くある誤解の話 - Qiita
 - https://qiita.com/angel_p_57/items/d7ffb9ec13b4dde3357d

著者紹介

mochiko / @mochikoAsTech

元 Web 制作会社のシステムエンジニア。技術書典で出した本がきっかけで、テクニカルライターの仕事を始めた。モバイルサイトのエンジニア、SIer とソーシャルゲームの広報を経て、2013 年よりサーバホスティングサービスの構築と運用を担当したのち、再び Web アプリケーションエンジニアとしてシステム開発に従事。「分からない気持ち」に寄り添える技術者になれるように日々奮闘中。技術書典 4,5,6 で頒布した「DNS をはじめよう」「AWS をはじめよう」「技術をつたえるテクニック」「技術同人誌を書いたあなたへ」は累計で 7,800 冊を突破。

- <https://twitter.com/mochikoAsTech>
- <https://mochikoastech.booth.pm/>
- <https://note.mu/mochikoastech>
- <https://mochikoastech.hatenablog.com/>

Hikaru Wakamatsu

表紙デザインを担当。

Shinya Nagashio

挿絵デザインを担当。

SSLをはじめよう

「なんとなく」から「ちゃんとわかる！」へ

2020-02-29/2020-03-01 技術書典 8 初版

著 者 mochikoAsTech

デザイン Hikaru Wakamatsu / Shinya Nagashio

発行所 mochikoAsTech

印刷所 日光企画

(C) 2020 mochikoAsTech